Universidad
Carlos III de Madrid

ESCUELA POLITÉCNICA SUPERIOR
DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

GRUPO DE SISTEMAS ELECTRÓNICOS DE POTENCIA (GSEP)

CONVERTIDORES DE POTENCIA MODULARES -ESTRUCTURAS DE CONTROL (específico)

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **David Ignacio Glaría Abril**
Tutores: **Antonio Lázaro Blanco - Universidad**
Jorge Rodríguez de Frutos - Empresa

Leganés, 25 septiembre 2018

Título: **CONVERTIDORES DE POTENCIA MODULARES-ESTRUCTURAS DE CONTROL (específico)**

Autor: **David Ignacio Glaría Abril**

Tutores: **Antonio Lázaro Blanco y Jorge Rodríguez de Frutos**

EL TRIBUNAL

Presidente:_____

Vocal:_____

Secretario:_____

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL SECRETARIO PRESIDENTE

RESUMEN

Se obtiene un método de sensado para los convertidores de potencia conmutados por modulación de ancho de pulso (PWM) de portadora triangular, que proporciona una medida exacta, rápida y limpia de la corriente media en cada periodo de conmutación, orientada al mejor funcionamiento del controlador del convertidor y válido para inversores de alterna trifásicos e implementable, con un coste razonable, mediante los recursos hardware y software disponibles en la actualidad.

Para ello, se estudian, analizan, se prueban en simulación, se evalúan y comparan varios métodos de sensado de la corriente de salida y se selecciona y propone el que se considera más adecuado y que resulte válido para los inversores trifásicos, que son el caso de aplicación más exigente por la complicación sobre la corriente que supone la interacción entre sus fases.

Se adjuntan listados de programas en lenguaje C que implementan para la herramienta de simulación PSIM los algoritmos de sensado síncronos que se estudian.

Palabras clave

Sensado de corriente; convertidor de potencia; modulación de ancho de pulso; PWM; portadora triangular; controlador del convertidor; inversor trifásico; PSIM; convertidor trifásico; sensado síncrono

ABSTRACT

A current sensing method for triangular carrier, pulse width modulation (PWM), switching power converters is obtained, that delivers an exact, fast and clean measure of the average current in each switching period, aimed at the best converter controller performance, valid for 3phase converters, and implementable, with a reasonable cost, by the hardware and software resources available nowadays.

To that aim, several output current sensing methods are studied, analyzed, tested in simulation, evaluated, and compared, and the method considered best fit, and valid for 3-phase inverters, which are the most demanding application cases, by the complication over the current caused by the interaction among their phases, is selected.

C-language program listings are enclosed which implement for the PSIM simulation tool the synchronous sensing algorithms which are studied.

Keywords

Current sensing; pulse width modulation; PWM; power converter; converter controller; 3phase converter; 3phase inverter; PSIM; synchronous sensing; triangular carrier

AGRADECIMIENTOS

Un folio no alcanza para mentar a todas aquellas personas que me han ayudado o influido durante mi formación en la universidad y, en concreto, en la culminación de este trabajo. Por ello, con gran esfuerzo de síntesis, quiero mencionar explícitamente mis agradecimientos especiales:

A mi tutor, Antonio, por ofrecerme este trabajo y orientarme. Sin él no habría sido posible.

A mi tutor de empresa, Jorge, siempre disponible para echar una mano, ayudar y enseñar con una sonrisa en la cara.

Al personal de Power Smart Control S.L., por su ayuda y amabilidad.

A mi familia, a todos ellos, ya que me han ayudado mucho en este trabajo con su apoyo y aguantándome en mis momentos de estrés. Y a mi sobrina Lucía, por su alegría contagiosa.

A mis compañeros, en especial a Sergio, que, aunque se haya ido lejos siempre me ha apoyado y echado un cable cuando lo necesitaba.

A la Universidad Carlos III, por el apoyo, soporte y los medios que puso en todo momento a mi disposición para proporcionarme una sólida formación como ingeniero electrónico industrial y ser una pieza clave en mi maduración personal.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Motivación del trabajo

La instrumentación de magnitudes eléctricas siempre ha representado un compromiso entre la potencia del muestreo empleada y la calidad y de la medida adquirida. Este concepto puede aplicarse de igual manera a los convertidores de potencia con una serie de matices que derivan del comportamiento ruidoso del mismo.

En un convertidor conmutado a alta frecuencia, supongamos controlado en corriente, ésta posee un rizado apreciable que pudiera estar montado sobre una componente fundamental de continua o, en diversas topologías de alterna, de 50Hz. A la hora de regular el sistema, se hace necesaria una o varias medidas de la magnitud a controlar, que frecuentemente es la media de la corriente a lo largo de un periodo de conmutación.

Si este sensado se produce por un muestreo a la frecuencia de conmutación, pueden existir errores de offset. Sin embargo, no se requerirá un filtrado posterior, ya que se medirá siempre el mismo punto de cada periodo.

Si se realiza un sobremuestreo, se capturará el rizado de conmutación. De no filtrarse, éste entrará en el regulador (analógico o digital) y será amplificado, generando problemas de estabilidad y empeorando la respuesta dinámica del mismo. En caso de filtrarse, se estaría añadiendo una pérdida de fase debida a la acción del filtro que, de nuevo, podría llegar a inestabilizar el sistema.

Es claro que un sensado de calidad, es decir, que proporcione al controlador una medida exacta, con el mínimo retardo y limpia, es imprescindible para un control de calidad, el cual, a su vez, incide muy notablemente en la calidad del convertidor.

Especial dificultad presenta el control de un inversor de tensión alterna trifásica, por la complejidad derivada de la gran interacción entre las tres fases. Dicho control requiere más que ningún otro un sensado de calidad, que también se ve dificultado por la complejidad de dicha interacción.

La motivación de este trabajo consiste en estudiar y analizar las distintas estrategias de sensado, disponibles o creadas para este propósito, y compararlas, de forma que se pueda extraer aquella que permita una correcta medición con una mínima caída de fase y que requiera un mínimo procesado o filtrado, así como que sea realizable, con un coste razonable, con los recursos hardware y software disponibles en la actualidad.

1.2 Objetivos

Este trabajo tiene el Objetivo General planteado en la motivación y varios Subobjetivos, u Objetivos subordinados.

1.- El **objetivo general** del presente Trabajo de Fin de Grado es obtener un método de sensado para los convertidores de potencia conmutados por modulación de ancho de pulso (PWM) de portadora triangular, que proporcione una medida exacta, rápida y limpia de la corriente media en cada periodo de conmutación, orientada al mejor funcionamiento del controlador del convertidor, que, como requisito específico explícito, sea válido para inversores de alterna trifásicos y que sea implementable, con un coste razonable, mediante recursos hardware y software disponibles en la actualidad.

Los subobjetivos son:

2.- Estudiar, analizar y comparar los métodos de sensado de la corriente de salida de un inversor y proponer el más adecuado o los más adecuados según las situaciones, siendo un requisito imprescindible que resulten válidos para los inversores trifásicos, que son el caso de aplicación más exigente por la complicación que supone la interacción entre sus fases.

3.- Implementar en simulación las estrategias aplicables, medir y comparar sus características de cara al objetivo de permitir un control de buena calidad.

4.- Seleccionar y proponer, de entre los métodos comparados, la mejor solución o soluciones para su objetivo de soportar un control de la máxima calidad.

5.- Validar el método de sensado propuesto, aplicándolo a las fases de un inversor trifásico por modulación de ancho de pulso PWM de portadora triangular, desarrollando un controlador sencillo que lo utilice y observando en simulación su comportamiento frente a perturbaciones.

6.- Comprobar la factibilidad de utilización de la solución seleccionada en microcontroladores, implementando sus algoritmos mediante unos programas en lenguaje C y utilizándolos en la simulación del objetivo 5 anterior.

1.3 Organización de la presente memoria

La presente memoria trata de recoger y explicar el proceso seguido en el presente trabajo, así como sus resultados, conclusiones, previsión de su posible impacto social, revisión de cumplimiento de objetivos y posibles trabajos de continuación.

Su esquema se recoge en el índice.

Comienza por una introducción en el Capítulo 1, en que se plantea la motivación del trabajo y, a partir de ella, se fijan su objetivo general y subobjetivos y se esquematiza su organización.

A continuación, se dedica el Capítulo 2 a plantear el contexto de trabajo: ubicar el problema mediante una breve explicación de los sistemas a los que los métodos de sensado estudiados se van a aplicar, los convertidores de potencia por modulación PWM, los inversores trifásicos, con especial incidencia en su control y en la problemática que plantea y en qué va a requerir del módulo de sensado. Se hace también una revisión del estado del arte de los métodos de sensado más utilizados actualmente y se añade alguna alternativa más que se propone para comparación, describiendo con más detalle su modo de funcionamiento y de implementación para la simulación y, por último, se establecen, previamente a la comparativa que se va a abordar, unos criterios de evaluación de su calidad, mediante unos indicadores cuantitativos, y su modo de medición.

En el Capítulo 3 siguiente, se documenta el proceso de diseño del inversor trifásico sobre el que se realizarán las simulaciones, la preparación y ejecución de su simulación y se aprovecha el resultado de dicha simulación para hacer un estudio en detalle de las corrientes implicadas en el sensado, que facilite la comprensión de su problemática y permita la posible invención de nuevos métodos de sensado o mejora de los existentes.

En el Capítulo 4, tras la implementación en PSIM del sistema de prueba y de los métodos de sensado a comparar, tras la definición de sus indicadores de calidad y de los

procedimientos para su obtención, se documentan las simulaciones realizadas del sistema trifásico en lazo abierto, los resultados obtenidos de ellas para los diferentes métodos de sensado y se presentan gráficos comparativos.

A partir de los resultados anteriores, en el Capítulo 5, se analizan y se elige la solución que se considera preferible y se propone.

En el Capítulo 6 se documenta la validación de la solución propuesta, mediante la evaluación de un controlador diseñado en base al método de sensado seleccionado.

En el Capítulo 7 se presenta el Marco Regulador aplicable al presente trabajo y en el Capítulo 8 el entorno socio-económico.

En el Capítulo 9 se recogen las conclusiones del estudio, se verifica el cumplimiento de los objetivos planteados en el Capítulo 1 y se proponen trabajos futuros de continuación.

En el Capítulo 10 se presenta la bibliografía consultada o relevante para el estudio

Y, finalmente, en los Apéndices A a F se adjuntan los programas en lenguaje C utilizados para implementación de la mayoría de los métodos de sensado estudiados.

2. CONTEXTO Y DESCRIPCIÓN DEL PROBLEMA

2.1 Introducción

En el presente capítulo se va a situar en contexto y describir el problema que se aborda en el presente trabajo.

Para ello, se va a realizar una breve introducción a los inversores, se va a presentar una clasificación de los diferentes tipos de inversores más utilizados en la actualidad, seguida de una breve presentación de los tipos principales de inversores que conducen al inversor trifásico fuente de tensión, que es sobre el que se ha definido realizar el estudio del sensado objeto del presente trabajo.

A continuación, se explicará con más detenimiento este inversor trifásico y su control, para poner claramente de manifiesto la importancia de un buen sensado.

2.2 Introducción a los convertidores de potencia inversores y clasificación

Un inversor es un convertidor de potencia DC-AC, que genera una tensión alterna a partir de tensión continua. Esto puede hacerse de muy variadas maneras. La más básica es aplicando alternativamente a la carga de salida los polos positivo y negativo de la tensión continua que suministra la energía, por medio de conmutadores, como se indica en la siguiente figura:

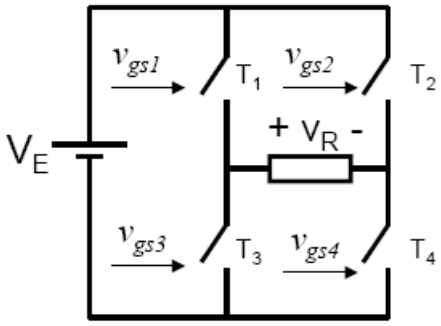


Fig 2.1 Inversor básico [1]

Alternativamente, se ponen en conducción los conmutadores T1 y T4 o T2 y T3, que aplican a la resistencia de carga la tensión V_R en un sentido o el opuesto, resultando en una tensión alterna, aunque cuadrada.

Normalmente, se desea que la tensión alterna de salida sea sinusoidal, lo que se consigue de varios modos. Por ejemplo, filtrando los armónicos superiores al fundamental de la onda cuadrada y dejando el fundamental, o modulando los tiempos relativos en que la tensión se aplica en un sentido o en otro. Los filtros que más se utilizan son los LC, porque no consumen potencia, con lo que no reducen la eficiencia (también pueden utilizarse RL, aprovechando la R de la carga, con lo que tampoco existe consumo de potencia adicional, pero el filtrado queda más dependiente de la carga que se conecte).

Dichos conmutadores T1 a T4 están implementados por dispositivos electrónicos, como transistores MOSFET o IGBT, gobernados por tensiones de control Vgs1 a Vgs4, respectivamente.

Dichas aplicaciones alternativas de potenciales de diferente signo sobre la carga pueden conseguirse hoy en día con facilidad, fiabilidad y economía gracias a los avances de la electrónica de potencia, guiados por los avances en los dispositivos electrónicos de conmutación de potencia y en los elementos de control (microcontroladores, FPGA, etc).

Por tanto, existen muchos tipos de convertidores diferentes. Aunque no es objeto de este trabajo tratar muchos de ellos, sí que parece conveniente presentar una clasificación de ellos según tres criterios diferentes, para situar el que se va a estudiar. Se presenta en la figura siguiente:

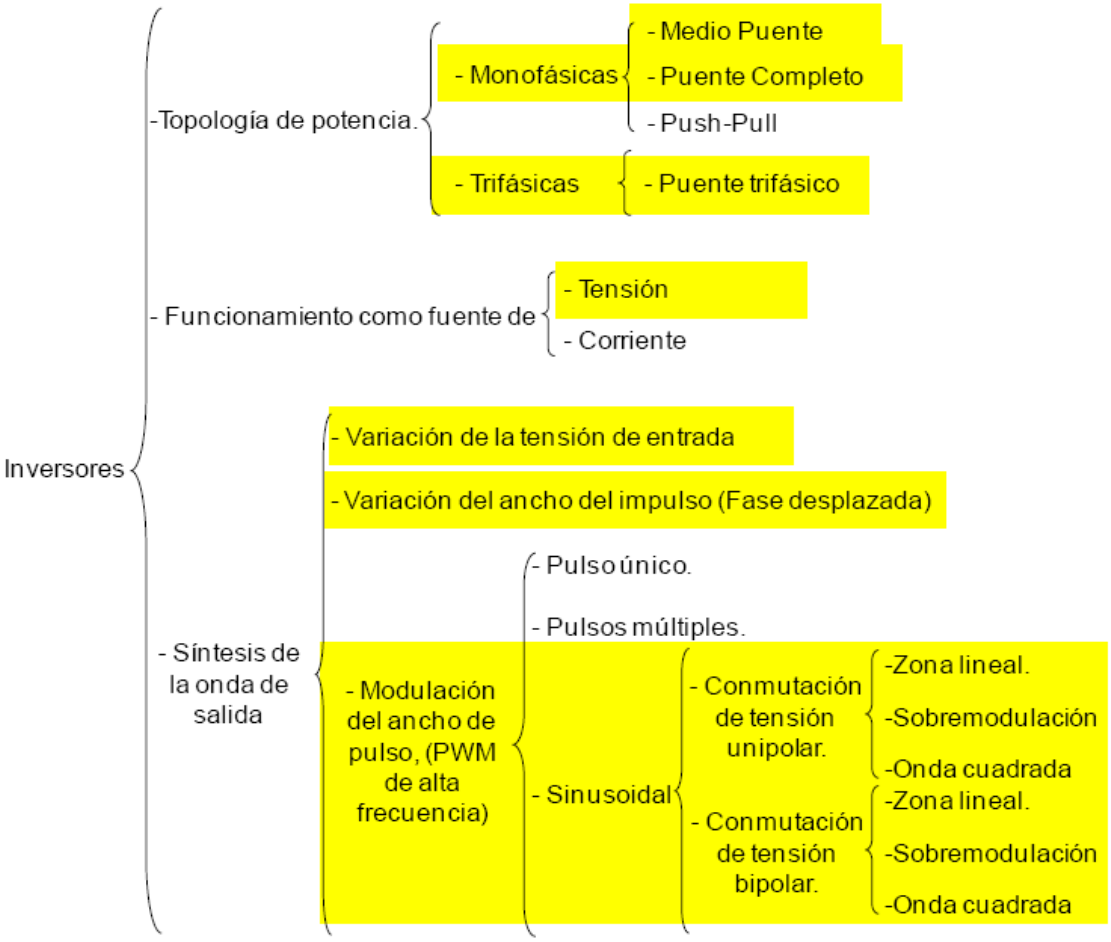


Fig. 2.2 Clasificación de inversores [1]

Dentro de la anterior clasificación, el inversor trifásico fuente de tensión que se va a estudiar es:

Por topología de potencia: **trifásico, con puente trifásico.**

Por tipo de fuente: **de tensión.**

Por síntesis de la onda de salida: **de modulación de ancho de pulso (PWM) sinusoidal, de tensión bipolar, trabajando en zona lineal.**

En los siguientes puntos se van a presentar los tipos de inversores cuya evolución ha conducido a él.

2.3 Inversor de onda cuadrada con variación del ancho del impulso

En la siguiente figura se presenta su esquema y su principio de funcionamiento:

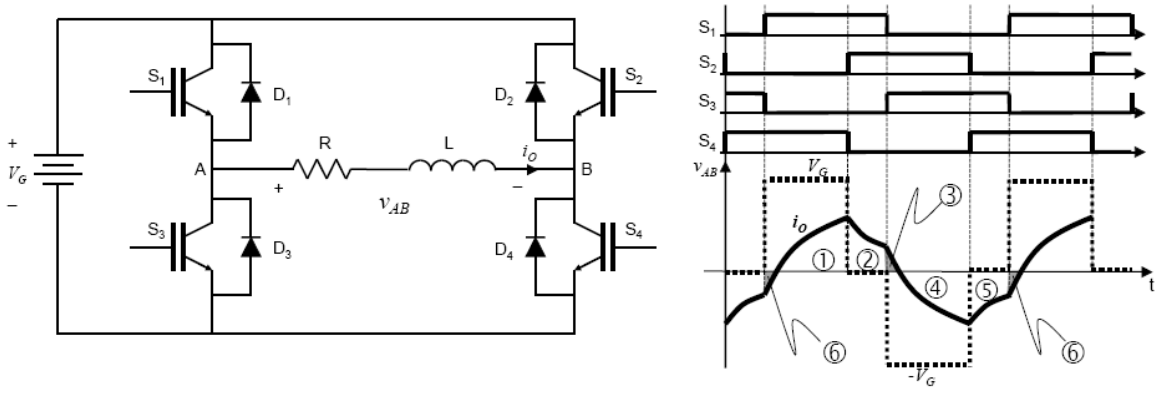


Fig. 2.3 Inversor de onda cuadrada con variación del ancho del impulso [1]

Como el principio básico de la figura 2.1, los conmutadores 1 a 4 de la figura 2.3 izquierda, aplican los polos de la batería de tensión continua a la carga R (a través de la inductancia de filtro L, para convertir la forma de onda cuadrada a sinusoidal), pero alternando del modo representado en la parte derecha de dicha figura 2.3, dejando unos tiempos en que la tensión aplicada entre A y B es 0 y la batería no entrega potencia. Modificando los tiempos relativos de unos u otros estados puede graduarse la amplitud de la tensión alterna aplicada a la carga, lo que permite igualmente efectuar una regulación de ella (control de su amplitud frente a variaciones, por ejemplo, de la tensión de la batería o de la carga).

2.4 Inversor con modulación PWM sinusoidal bipolar de puente completo

El esquema de la etapa de potencia es el mismo de la figura 2.3 izquierda, salvo que en algunos casos se añade un condensador en paralelo con la salida (que queda, por tanto, en paralelo con la carga) para mejorar el filtrado de la tensión de salida.

La diferencia fundamental con el inversor del punto anterior radica en el modo (más precisamente, la cadencia) de gobernar las conmutaciones de los conmutadores electrónicos. En este caso, las tensiones de gobierno aplicadas a los conmutadores se generan por una modulación de ancho de pulso, PWM ("Pulse Width Modulation"), que consiste básicamente en hacer el ancho del pulso de una señal rectangular proporcional a la señal moduladora. Esto se lleva a cabo mediante comparación (mediante el comparador electrónico representado por el triángulo) de una moduladora de baja frecuencia (Vmoduladora) y una portadora triangular (Vportadora) de alta frecuencia, como se explica en la figura siguiente:

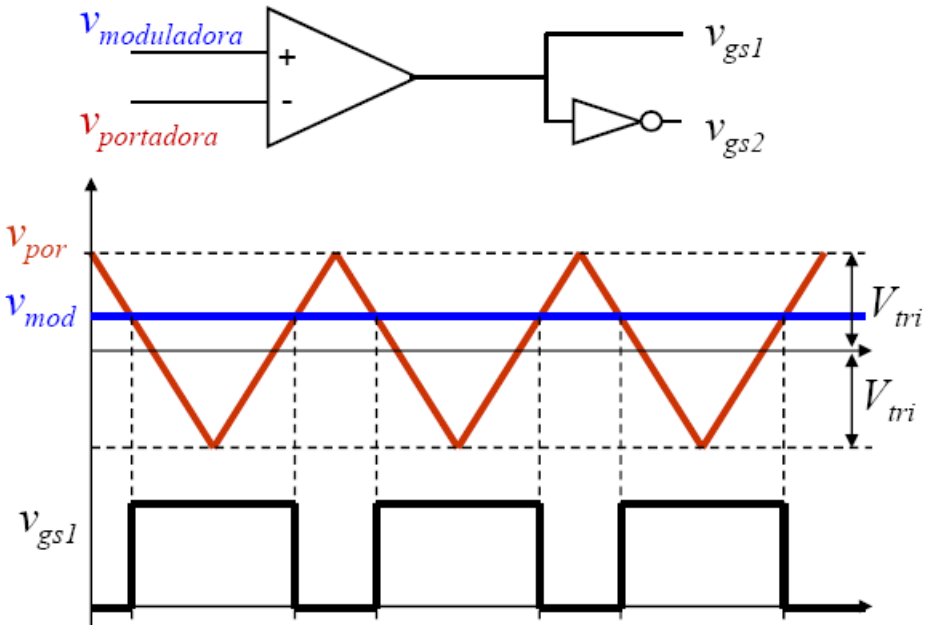


Fig. 2.4 Generación de las tensiones de gobierno de los conmutadores por modulación PWM, mediante comparación de una moduladora y una portadora triangular [1].

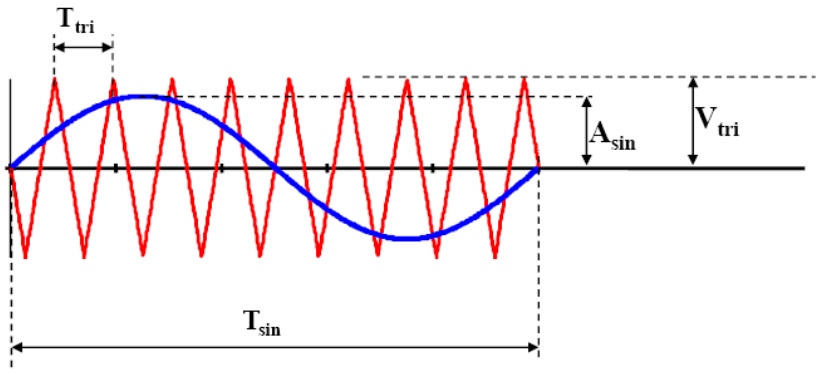
La tensión Vgs1 se aplica simultáneamente a Vgs4 y la opuesta, Vgs2, a Vgs3.

Con este esquema, el ancho de los pulsos en que se aplica tensión positiva respecto al total del ancho del ciclo determina una corriente media en cada periodo de modulación (el periodo de la portadora triangular) proporcional a la tensión moduladora.

Haciendo que la moduladora sea una tensión sinusoidal, se consigue que la tensión de salida generada, obtenida por promediado de la tensión PWM de salida (promediado que se produce en el filtro RL o RLC de salida) sea la sinusoidal deseada.

Dicho filtro, además, elimina los armónicos de la portadora triangular que estarían presentes en la tensión de salida. Al ser la frecuencia de esta portadora muy alta, en relación a la frecuencia de la moduladora, la eficacia de eliminación del filtro es también muy alta. Basta para ello con componentes de valores L y C muy inferiores a los necesarios para los inversores de onda cuadrada.

La figura siguiente explica el proceso de generación de esta sinusoidal y dos parámetros importantes que la definen:



Índice de modulación en amplitud

$$m_a = \frac{A_{\sin}}{V_{tri}}$$

Índice de modulación en frecuencia

$$m_f = \frac{T_{sin}}{T_{tri}} = \frac{f_{tri}}{f_{sin}}$$

Fig. 2.5 Modulación PWM sinusoidal y sus parámetros [1].

Como se indica, el índice de modulación en frecuencia es la relación de la frecuencia de la portadora triangular a la de la sinusoide moduladora. A mayor frecuencia de la portadora, más fácil y económico resulta su filtrado, aunque como contrapartida, exige mayor velocidad a los componentes del inversor. Fue precisamente la aparición de componentes conmutadores de mayor velocidad, menores pérdidas y menor coste lo que permitió pasar del inversor de onda cuadrada a éste de modulación PWM.

El índice de modulación en amplitud, $\mathbf{m_a}$, es la relación de la tensión de pico de la moduladora sinusoidal a la tensión de pico de la portadora triangular. Este índice es muy importante, pues permite graduar la amplitud de la tensión sinusoidal de salida y, por tanto,

efectuar una regulación de ella (control de su amplitud frente a variaciones, por ejemplo, de la tensión de la batería o de la carga).

En la figura siguiente se aprecia mejor el proceso de modulación, la tensión VA – VB, aplicada al filtro de salida y su tensión media, sinusoidal (la sinusoidal azul en el gráfico inferior), trasladada por el filtro a la carga:

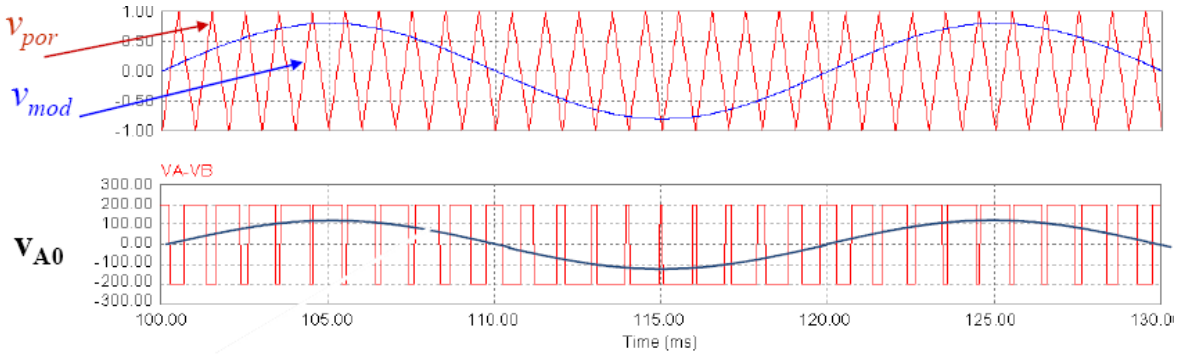


Fig. 2.6 Modulación PWM: tensión VA – VB, aplicada al filtro de salida y su tensión media, sinusoidal (azul inferior), trasladada por el filtro a la carga [1]

2.5 El inversor trifásico fuente de tensión VSI

El inversor trifásico de modulación de ancho de pulso PWM es una extensión del anterior, en el que se generan 3 tensiones alternas desfasadas 120º, para entregar a cada una de las fases.

En la siguiente figura se presenta el esquema de su etapa de salida y de la modulación PWM que genera sus señales de control:

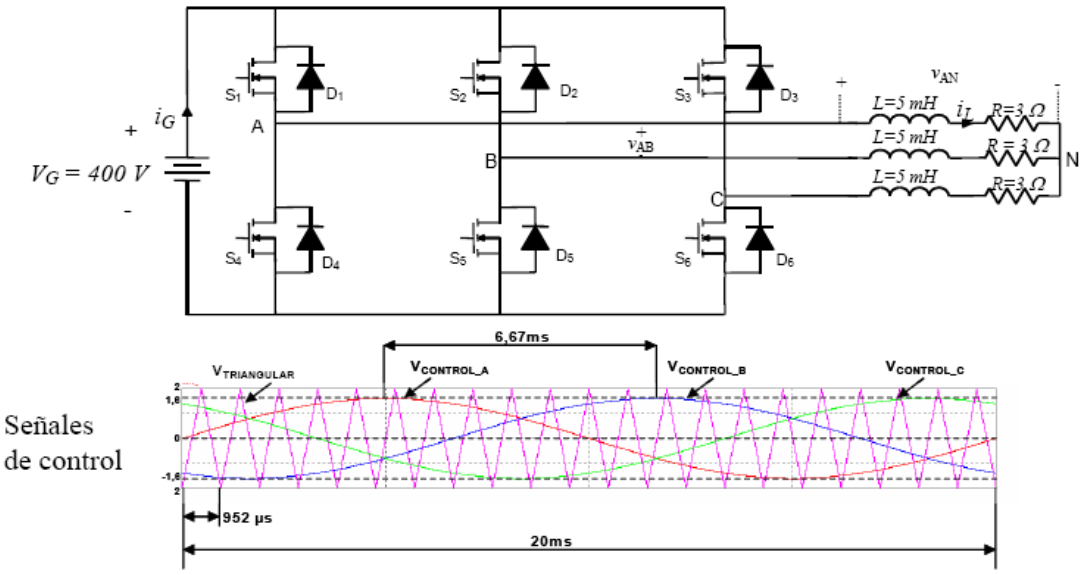


Fig. 2.7 Inversor trifásico [1]

En la figura se observa cómo las salidas se entregan a la carga. Como la tensión alterna produce un rizado importante, se colocan en serie unas inductancias de filtro que la filtran. Es común además conectar unos condensadores desde cada fase a un punto común que hace de neutro, para mejorar el filtrado estableciendo un filtro de segundo orden con mejor atenuación.

La tensión del neutro, que resulta de la media instantánea de las 3 fases produce un acoplamiento importante entre ellas hacia la corriente por cada salida.

2.6 Control del inversor trifásico VSI

El control del inversor trifásico VSI, fuente de tensión persigue mantener regulada su tensión frente a las posibles perturbaciones. Las más importantes son variaciones de la tensión de alimentación continua Vdc y de la carga.

Las perturbaciones producen un cambio rápido de las corrientes de salida de los conmutadores, que se integran en los condensadores del filtro de salida para ir produciendo, más lenta y progresivamente, una variación de la tensión de salida. Por eso, un control de

corriente, que detecte rápidamente y corrija sus variaciones, va a resultar en una corrección más rápida de las tensiones de salida que si se espera a detectar los cambios de tensión. Precisamente para dicha detección rápida es importante que los sensores produzcan el menor retardo posible.

Pero, al tratarse de una fuente de tensión, es la tensión lo que debe controlarse. Esto suele hacerse en una configuración multilazo en que un lazo interno de corriente, rápido, es controlado (en cascada) por un lazo externo de tensión, más lento, según se ilustra en la siguiente figura:

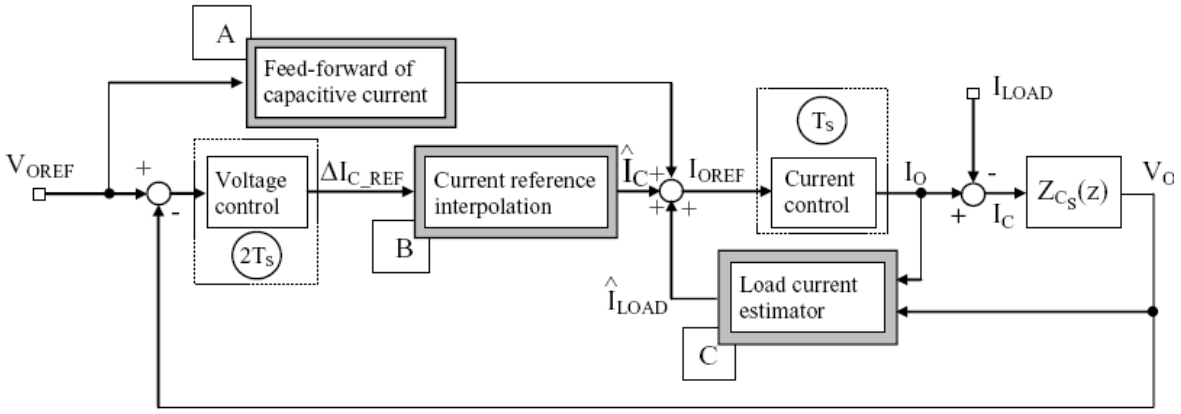


Fig. 2.8 Control multilazo del inversor trifásico [1]

2.7 Sensado de la corriente de salida para control del Inversor VSI

Como se ha explicado previamente, un elemento fundamental para un buen control es una buena medida de la magnitud a controlar, en este caso, la corriente de salida del inversor.

Al retardo inevitable de un periodo de modulación inherente al PWM, se añade el posible retardo del sensor de medida de la corriente de salida.

Por eso es muy importante que el método de sensado añada el mínimo retardo a la medida que extrae de la corriente de salida y entrega al controlador.

En el capítulo siguiente se repasarán los métodos más habituales o que se han considerado más prometedores para realizar dicha medida y sus diferentes características.

Para el caso trifásico, el sensado por muestreo síncrono en los extremos de la portadora es muy complicado de analizar teóricamente. En el capítulo 3 se estudiará en detalle sobre una simulación las posibilidades con las magnitudes eléctricas disponibles.

Dado que van a estudiarse y compararse varios métodos de sensado, es importante definir con precisión cuál es la magnitud a proporcionar por ellos. Teniendo en cuenta el proceso de modulación PWM, la magnitud que interesa medir, para que el controlador la regule, es la **media de la corriente en cada periodo de modulación**.

Por tanto, para juzgar la calidad de cada método de sensado, se valorará en qué grado dicho método proporciona una medida de dicha media de corriente con la mayor exactitud y la mayor antelación.

2.8 Estado del arte de los métodos de sensado y alternativas a estudiar

En el presente punto se van a repasar los métodos de sensado utilizados más frecuentemente y se van a presentar y describir los métodos que han parecido más prometedores, a estudiar en este trabajo.

Los métodos más utilizados actualmente son los siguientes:

a. Sensado **a frecuencia de conmutación**: toman medidas de la corriente en ciertos instantes de cada periodo de modulación buscando derivar de ellas una buena estimación de la corriente media en dicho periodo.

b. Estrategias de **corrección de error** en régimen **permanente del sensado a frecuencia de conmutación**: Añaden una ganancia extra en el sensado para compensar este error, que se produce a en algunas circunstancias, como cuando, por varias posibles causas, aparece un error de sincronización que consiste en que los instantes de muestreo reales no coinciden con los deseados. Por varios procedimientos, determinan la diferencia de amplitud en la señal sensada respecto a la correcta y aplican una ganancia correctora, del valor necesario para recuperar una aproximación al valor correcto.

c. **Multisampling**: Consiste en tomar varias muestras en cada periodo de conmutación, generalmente equiespaciadas en el tiempo, y hacer un promediado de todas ellas.

De este modo, la medida resultante es menos sensible al error de alguna de las muestras. Además, al estar las muestras distribuidas por todo el periodo, cada una representa el valor de la corriente en una zona diferente del periodo, de modo que su media se aproxima más a la media de la corriente en todo el periodo.

Otra ventaja es que, al caer las muestras en rampas de corriente de diferente pendiente, un posible offset en los instantes de muestreo produce en las muestras errores de signo contrario que se compensan al hallar su media.

Sin embargo, el multisampling tiene el inconveniente de presentar al hardware una exigencia mayor en cuanto a velocidad de adquisición, por lo que puede aumentar su coste.

Por otro lado, al contribuir a la medida muestras más antiguas, su media representa el valor de la corriente en instantes más pasados, lo que equivale a un retardo mayor de la medida.

d. Estrategias de **filtrado para multisampling**: si se intenta reducir el retardo quedándose sólo con muestras más recientes, la medida producida queda con un mayor rizado residual. Un modo de reducirlo es aplicarle un filtrado, pero esto tiene el peligro de que el filtro añada a su vez un desfase a la medida que perjudica al sistema de control: puede desestabilizarlo o restringir gravemente .su ancho de banda y, por tanto, su velocidad de respuesta.

Para analizar los métodos de sensado y pensar en otros posibles, es necesario profundizar más en la problemática del sensado, lo que se va a hacer en los siguientes párrafos.

Como se ha comentado, para conseguir un buen control del inversor, lo que se necesita es una medida de la corriente lo más exacta y temprana posible.

En la figura 2.9 siguiente se presentan dichas corrientes en la L y en la R de la carga en una de las fases, en un zoom de dos periodos de modulación, con las tensiones en las otras dos fases anuladas para simplificar.

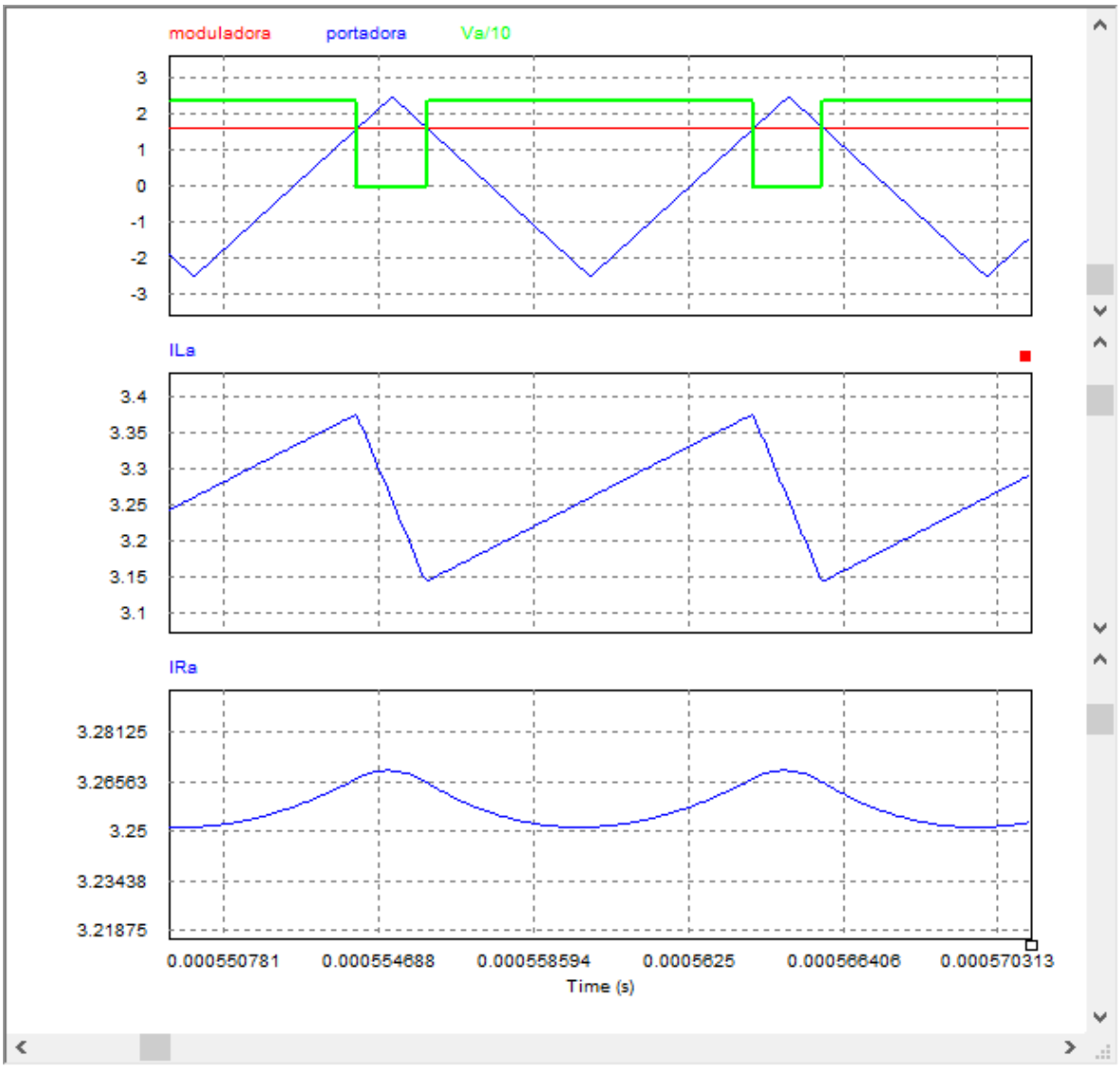


Fig. 2.9 Corrientes en la L del filtro de salida y en la R de la carga en monofásico

Puede apreciarse cómo la corriente en la L i_L tiene un rizado considerable (Vpp alrededor el 30% del valor medio) y mucho menor (alrededor del 3%) en la carga R. Pero, todavía más importante, se aprecia cómo la corriente en la carga está bastante desfasada respecto a la corriente en la L. También se observa que la corriente por la L describe trayectorias muy rectas entre las conmutaciones, lo que se debe a que su constante de tiempo con la carga es mucho mayor que el periodo de modulación.

Dado que la corriente trasladada a la carga por el filtro esté retrasada respecto a la corriente entregada al filtro por el inversor, interesa más medir la corriente entregada al filtro, que circula por su inductancia serie. La dificultad es el gran rizado que tiene esta corriente, proveniente de los grandes saltos de tensión que la etapa de potencia aplica a la inductancia.

Como se sabe,

$$\frac{\Delta i_L}{\Delta t} = \frac{V_L}{L} \tag{2.8.1}$$

y esa v_L es la tensión PWM con pulsos de Vdc

El filtro LC de salida suaviza en gran medida ese rizado, de modo que la parte que llega a la carga es perfectamente admisible, pero el controlador necesita una señal más limpia para funcionar con calidad.

Para limpiar el rizado de la iL y extraer su valor medio, el primer método en que se piensa es el uso de **filtros analógicos**. El problema con estos filtros es que inevitablemente añaden un retardo a la señal que es muy perjudicial para el control.

En todo caso, en este trabajo se estudiarán tres filtros analógicos, que se van a describir en un punto posterior.

Otro método de sensado que parece muy adecuado para el objetivo sería un **integrador analógico de corriente reiniciado** (puesto a 0) al principio de cada periodo de modulación. Este método proporcionaría, justamente, la medida que se busca, tal como ya se ha definido al final del punto 2.7: la media de la corriente en el periodo de modulación, por lo que también se va a probar y, además, es el que se va a utilizar como **referencia de amplitud** respecto a la que calcular los errores de amplitud de todos los demás métodos de sensado.

Este integrador sería, en principio, el método de sensado ideal, ya que proporciona su medida al final de cada periodo de modulación, que, a primera vista, es el instante más temprano en que podría disponerse de ella. Pero conviene reconsiderar esto más detenidamente:

En general, el valor de la media de la corriente en un periodo no puede conocerse hasta que dicho periodo ha terminado. Esto es así, en efecto, cuando dicha corriente puede seguir una

evolución arbitraria. Sin embargo, en el caso particular del inversor, dicha corriente es el efecto de un proceso (de modulación PWM y filtrado) de ley conocida, y podría intentarse predecirla a partir de medidas anteriores al fin del periodo. Esto es lo que se hace en varios métodos de sensado denominados “síncronos”, por estar sincronizados con la modulación, que toman varias muestras de corriente instantánea en ciertos instantes del periodo de modulación y las procesan (generalmente, calculan su media). Se basan en que la corriente por la inductancia sigue una trayectoria simétrica respecto a su valor en los instantes de los vértices de la portadora triangular. Resultan muy prometedores, por lo que en el presente trabajo se han examinado un buen número de métodos de sensado síncronos, que se van a presentar en el punto 2.10 próximo.

Otro modo de filtrado posible de la corriente es mediante los **filtros digitales.** De ellos, existen dos grandes grupos: FIR (Finite Impulse Response) e IIR (Infinite Impulse Response). Los FIR calculan una salida como una suma ponderada (por unos coeficientes) de las entradas anteriores; y, los IIR, como una suma ponderada de las entradas y también de las salidas anteriores.

En general, permiten implementar discretizadamente los filtros analógicos, pero sin superarlos en prestaciones. Por esto, habría convenido considerarlos para estudio, si alguno de los filtros analógicos se eligiera como solución al sensado, pero, como se verá más adelante al examinar los resultados, no es el caso.

Por otro lado, el método de sensado síncrono por media deslizante de determinado número de muestras podría considerarse un caso particular (bastante trivial) de filtro digital FIR, con todos los coeficientes iguales, pero este método ya va a estudiarse dentro del grupo de los métodos síncronos. Por tanto, en el presente trabajo, no se van a estudiar estos filtros digitales.

A los distintos métodos de sensado que se estudian en el presente trabajo se les ha dado nombres que se ha intentado que sean algo descriptivos, pero cortos, para utilizarlos en las tablas de resultados y de comparativas. Estos nombres se indican en su respectiva descripción en los puntos siguientes.

2.9 Sensado con filtros analógicos

Como se ha comentado, intentan extraer de la corriente por la inductancia, ensuciada por un rizado apreciable, su valor medio (pero no en un periodo definido). Interesa buscar tipos de filtro que, por un lado, entreguen un valor exacto de la corriente media y, por otro, añadan el mínimo retardo posible.

Cuanto mayor sea el orden de los filtros, más atenúan el rizado, pero más retardo añaden.

El primero que va a probarse es un filtro analógico paso bajo de orden 2 ofrecido por PSIM, con el "damping ratio" de 0.7 que propone por defecto y frecuencia de corte 10KHz, buscando una fuerte atenuación a la frecuencia de modulación y que deje un ancho de banda suficiente para respuesta al control. Se ha denominado "**LPFSimple10K**".

El segundo que va a probarse es una combinación en cascada de los dos filtros analógicos paso bajo ofrecidos por PSIM, ajustados con un "damping ratio" de 0.7 (el que propone por defecto) y frecuencia de corte 20 KHz. Se ha ajustado a una frecuencia de corte superior, porque va a tener doble pendiente de atenuación y producirá una atenuación a la frecuencia de modulación parecida al anterior. Se ha denominado "**LPFDoble20K**".

El tercero, buscando una respuesta lo más fiel posible, va a ser un filtro de Bessel de orden 4 y de frecuencia de corte 40 KHz. Se diseñó inicialmente con MATLAB para una frecuencia de corte de 4 KHz, cuando se efectuaban simulaciones con frecuencia de modulación de 10 KHz, y, posteriormente, se convirtieron sus coeficientes a la nueva frecuencia de corte de 40 KHz cuando la frecuencia de modulación se cambió a 100 KHz.

Los coeficientes (de las potencias de s en numerador y denominador de la función de transferencia) que lo definen son:

B

0 0 0 0 3.98987636875274e+17

A

0.0001 78.51317405 27739433.25 5.08179E+12 3.98988E+17

Se ha implementado en la simulación en PSIM mediante el elemento “s-domain Transfer Function”, H(s), del PSIM configurándolo con dichos coeficientes. Se ha denominado “**Bessel**”.

2.10 Muestreadores síncronos

Como se ha adelantado, los muestreadotes asíncronos son métodos de sensado de la corriente por la inductancia sincronizados con la modulación, que toman una o varias muestras de corriente instantánea i_L en ciertos instantes del periodo de modulación y las procesan.

Están basados en el conocimiento de la respuesta de la i_L a la tensión modulada por anchura de pulsos PWM que se aplica a L. Como se ha comentado anteriormente y se observa en el gráfico segundo de la figura 2.9 para una de las fases con las otra dos fijas, i_L sigue rampas muy aproximadamente rectas entre las conmutaciones de la v_L, que, a su vez, alterna entre V_{DC}.y masa. Como las conmutaciones son también (por su modo de generación por comparación de la portadora triangular con la moduladora lenta) simétricas respecto al momento de la cresta de los triángulos (y también de los valles), las rampas también lo son.

Cuando las otras dos fases también son moduladas en trifásica, aparece una interacción entre las fases que complica el análisis, pero, si son moduladas por el mismo procedimiento y con la misma portadora, sigue manteniéndose la simetría respecto a los vértices de la portadora.

Aprovechando este conocimiento de la simetría de la corriente instantánea, pueden desarrollarse métodos de sensado que obtengan su valor medio.

Por ejemplo, el más evidente y simple, es tomar una muestra de la corriente instantánea justo en el instante de un vértice de la portadora triangular (como se ilustra en la figura 2.11 posterior). Por la simetría de la corriente instantánea, el valor medio de la corriente debe coincidir con dicha muestra. Otras alternativas son tomar más muestras simétricas respecto al vértice y promediarlas, lo que otra vez debe coincidir con el valor medio en el periodo de modulación. La ventaja de tomar más muestras por periodo es que se reduce la sensibilidad frente a errores de una sola de ellas.

Otro aspecto importante del procedimiento de generación de la PWM a tener cuenta para el sensado es la **cadencia de actualización** de valores de la moduladora: Cuando la modulación es analógica, la moduladora puede afectar a la posición en el tiempo de la conmutación en cualquiera de las rampas, de subida o de bajada, de la portadora triangular. Cuando la modulación es digital, lo que es lo más frecuente en inversores controlados por un controlador inteligente, como microcontrolador, DSP o FPGA, la actualización del valor de la moduladora puede efectuarse antes del comienzo de cada rampa (lo que se denomina actualización doble) o sólo antes del comienzo del periodo de modulación, es decir, antes de la rampa de subida (actualización simple). El tipo de actualización puede hacer que algunos métodos de sensado sean o no ventajosos respecto a otros.

El modulador PWM que se ha implementado en PSIM, con el que van a realizarse todas las pruebas, es de tipo analógico y produce una actualización doble.

En el presente trabajo se van a estudiar varios métodos de sensado síncronos, la mayoría basados en la anterior idea. En los siguientes puntos se van a describir estos diferentes métodos, indicando sus posibles ventajas e inconvenientes a priori y en los capítulos 4 y 5 se presentarán los resultados obtenidos al probarlos en simulación.

2.10.1 Integrador de corriente reiniciado, sin o con “Second Order Hold”

El método de sensado **integrador analógico** de corriente reiniciado (puesto a 0) al principio de cada periodo de modulación [6], ya mencionado anteriormente, es un método síncrono, dado que necesita sincronizarse con la modulación para reiniciarse justo al principio de cada periodo de modulación y entregar la media (suma acumulada dividida entre número de muestras) de los valores instantáneos de la corriente justo al final de dicho periodo. Tiene el inconveniente de que requiere hardware adicional y su uso no es muy

extendido [2]. Tiene las ventajas de que da el valor de media más exacto y de que ahorra procesamiento en el microcontrolador (el hardware adicional actúa como un procesador analógico externo).

Inicialmente, se implementó con elementos de la librería del PSIM, pero, como tenía un funcionamiento algo deficiente (por ser bastante críticos los instantes de comienzo y fin del periodo), se ha implementado en un bloque en C, con lo que se consigue gran exactitud (comprobada con la media de la corriente en postproceso en el SimView).

El código es muy simple y se lista en el apéndice A. Lo único especial que cabe mencionar es la siguiente línea, que consigue identificar con precisión los instantes de salto de periodo:

```
if( (sSigT_d - t) < (t + delt - sSigT_d ) )
```

La variable `sSigT_d` contiene el tiempo del instante siguiente esperado, pero no puede compararse directamente con la `t` recibida, que contiene el tiempo de simulación del presente paso de simulación, porque pueden no coincidir (las dos son de tipo flotante "double"). La expresión utilizada permite localizar el instante de tiempo de simulación más próximo al instante esperado.

Se ha denominado "**IntegradorNoSOH**"

Se estudia también otra versión de éste, denominada "**Integrador**", que se obtiene añadiéndole un "**Second Order Hold**" (**SOH**) (retenedor de segundo orden), que, para suavizar la transición entre los puntos por un escalón, proyecta desde el último punto hacia adelante una curva que sigue las derivadas primera y segunda correspondientes a los puntos anteriores. Para control discreto, no suponen diferencia, porque las trayectorias de las dos curvas coinciden en todos los puntos iniciales de los escalones, pero son útiles para control analógico, por aportar transiciones más suaves que desestabilizan menos a los controladores.

Por el mismo motivo, este SOH se ha utilizado en este trabajo también con todos los métodos de sensado que generan escalones. En algún caso en que se ha apreciado que

pueden ser contraproducentes, se ha dejado también la versión sin SOH, que se ha denominado …NoSOH.

Evidentemente, estos algoritmos deben funcionar en tiempo real, no en postproceso. Por eso, no vale interpolar entre puntos ya conocidos, ya que el punto próximo en tiempo real es desconocido.

Inicialmente, se desarrolló un “First Order Hold”, que rellena las transiciones con rampas rectas que siguen la derivada de los puntos anteriores, pero se vio que, por poco esfuerzo de programación más, el SOH tiene un comportamiento más satisfactorio.

La implementación del SOH se ha realizado también con un bloque de código C cuyo programa se lista en el apéndice B.

Para solucionar el problema de que el PSIM, tras un flanco de escalón, no se estabiliza inmediatamente, sino que tiene cierta transición, se ha añadido al comienzo del disparo la condición de que el nuevo valor se repita, como confirmación de que ya se ha estabilizado.

Además, para evitar algunas irregularidades no convenientes observadas, la curva se ha restringido de modo que no pueda ir más rápido que la rampa del FOH y que no pueda retroceder de su sentido inicial.

Aunque al final ha acabado no usándose, se ha dejado en el mismo bloque C una patilla de salida que da las rampas del “First Order Hold”, “FOH”.

En la figura siguiente se ilustra el comportamiento de este SOH:

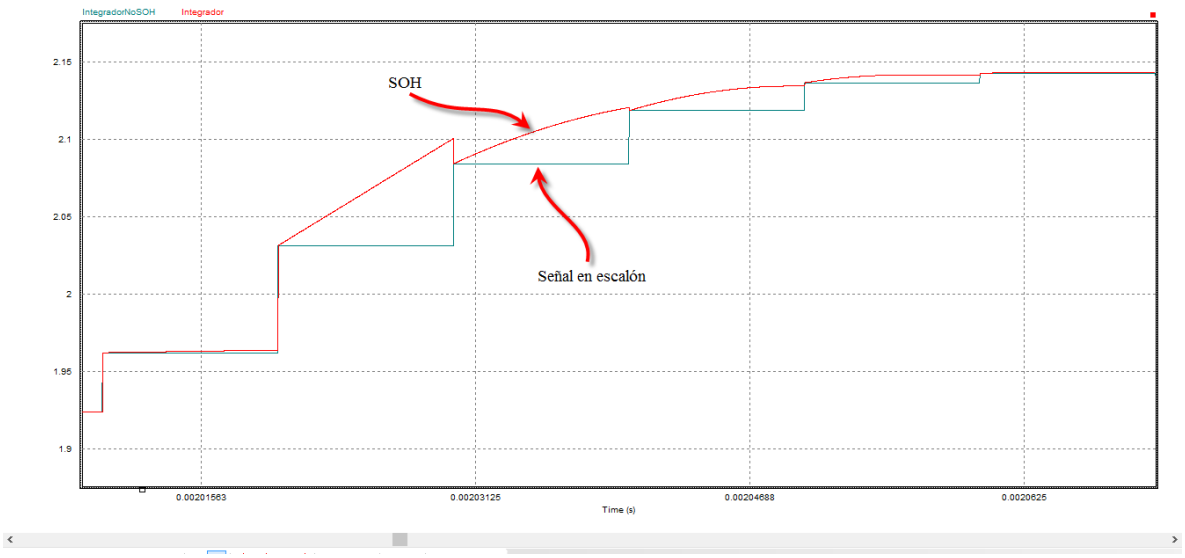


Fig. 2.10 Comportamiento del SOH suavizando una señal en escalón

Puede observarse cómo la curva marcada SOH suaviza la transición entre los puntos de la izquierda de las horizontales de la señal en escalón (como podría ser la del integrador o cualquiera de las muchas que entregan los muestreadores síncronos). El algoritmo la genera a partir de las derivadas primera y segunda en el periodo último. Cuando la curva llega al instante del nuevo valor, salta en vertical a dicho nuevo valor, para asegurar la mayor fidelidad con las medidas reales (podía haberse diseñado una transición más suave desde ese punto hacia el siguiente, pero se ha preferido la fidelidad a la suavidad).

En el escalón anterior al marcado con la flecha, se observa cómo la trayectoria es una recta en lugar de una curva. Eso se debe a que la derivada segunda habría producido una curva que rebasaría a la rampa del FOH, por lo que el algoritmo deja en su lugar esta rampa. El escalón anterior a la rampa no está casi suavizado, porque es el primero de la serie de escalones de subida y la derivada primera anterior era casi 0.

En general, ya en el resto de escalones, puede observarse cómo el algoritmo predice bastante bien, a partir de los anteriores, el valor próximo de la señal y genera una curva bastante suave que pasa cerca de todos ellos.

2.10.2 Muestreador en los vértices de la portadora triangular, sin o con SOH

Toma dos muestras por periodo de la corriente instantánea por la inductancia: en los instantes de los vértices (crestas y picos) de la portadora triangular, y entrega sólo el valor de la última muestra tomada. Se ilustra en la siguiente figura:

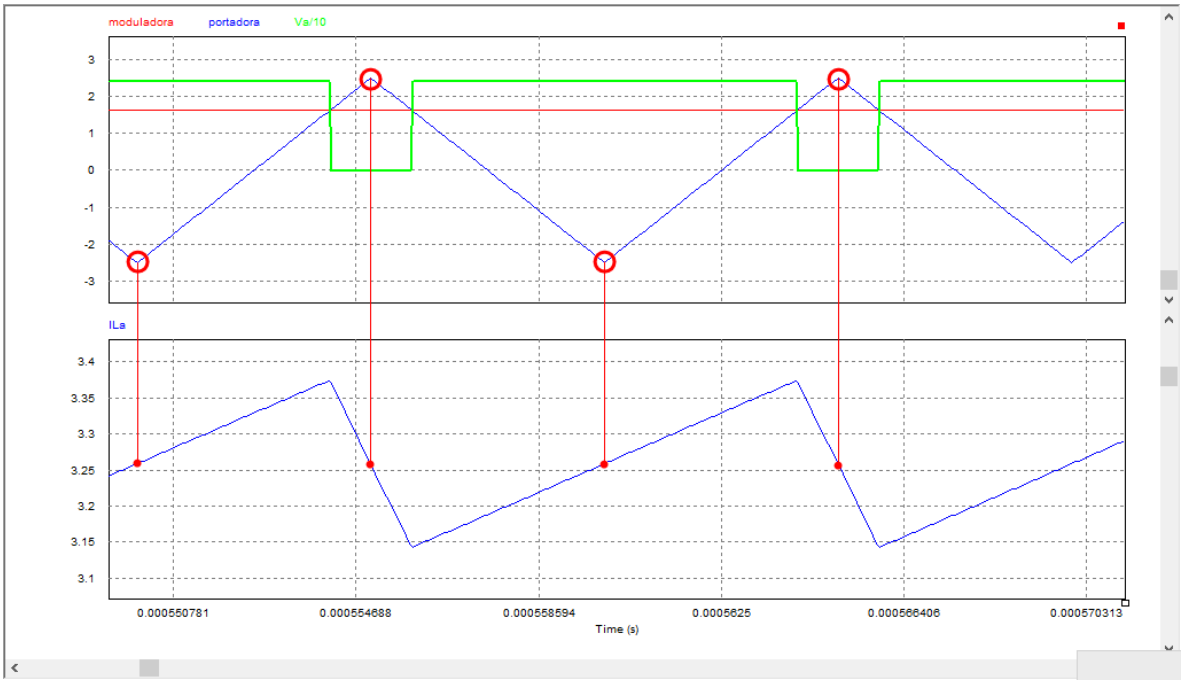


Fig. 2.11 Muestreo en los vértices de la portadora triangular

Cuando la corriente es simétrica respecto a esos puntos (como ocurre con mucha aproximación cuando la constante de tiempo de la carga por la inductancia es mucho mayor que el periodo de modulación), cada uno de esos valores coincide con el valor medio en todo el tiempo de un periodo centrado en el vértice.

A priori, tiene la ventaja de que adelanta medio periodo de modulación la disponibilidad de la media respecto a cualquier método que la proporcione sólo al final de los periodos.

Tiene el inconveniente de que, si la pendiente de la corriente es grande, el valor obtenido puede ser muy sensible a errores de sincronización. También puede tener otro inconveniente en algunos casos de simetría imperfecta: que los valores que dé en las crestas sean ligeramente diferentes del que dé en los valles y la señal que entregue tenga cierta alternancia residual. Cuando esto ocurre, el SOH puede ser perjudicial, porque puede amplificar dicha alternancia. Por esto, se dan dos versiones de este método: una sin SOH,

que se ha denominado "**Mdesliz1y2_2_NoSOH**" y otra con SOH, denominada "**MDesliz1y2_2**".

Se ha implementado en un bloque C, cuyo código se lista en el apéndice C, que se comentará en un punto posterior.

2.10.3 Muestreador en las crestas de la portadora triangular

Toma en cada periodo sólo una muestra de la corriente instantánea por la inductancia en el instante de la cresta de la portadora triangular, como se ilustra en la siguiente figura, y lo entrega tan pronto está disponible:

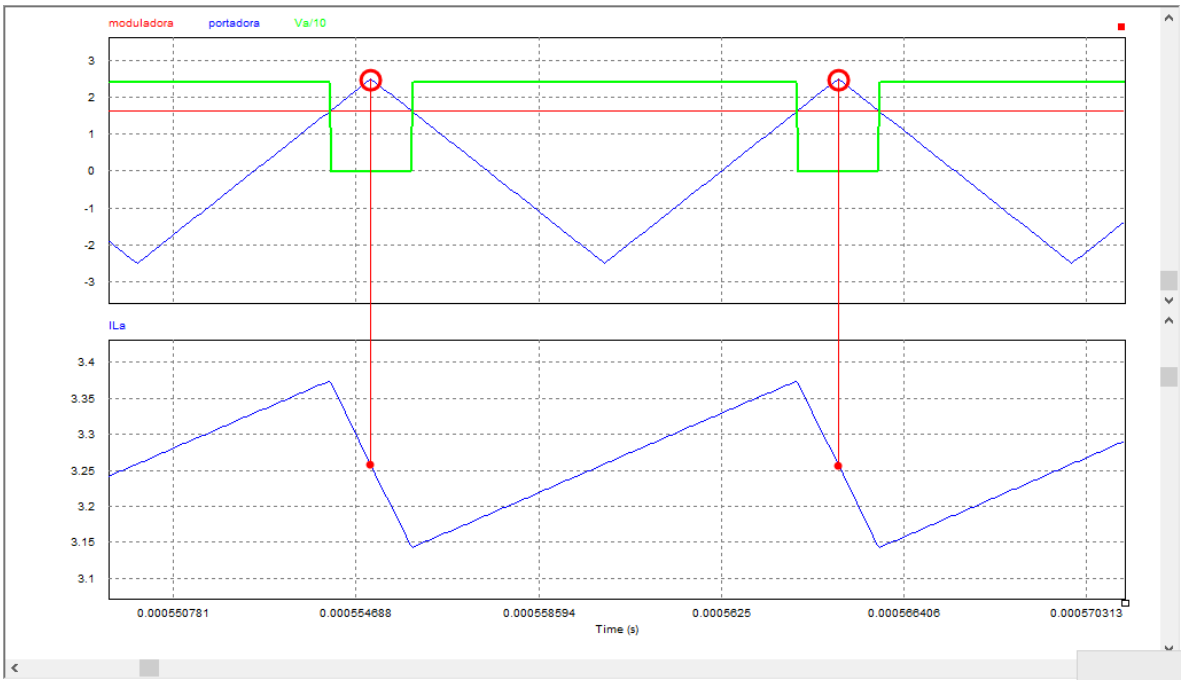


Fig. 2.12 Muestreo en las crestas de la portadora triangular

Cuando la corriente es simétrica respecto a ese punto (como ocurre con mucha aproximación cuando la constante de tiempo de la carga por la inductancia es mucho mayor que el periodo de modulación), ese valor central de la corriente coincide con su valor medio en todo el periodo.

A priori, si la actualización del PWM es simple, sólo al comienzo del periodo, tiene la ventaja de que proporciona su medida de la media sólo medio periodo de modulación después de la actualización, adelantando en medio periodo de modulación su disponibilidad

respecto a cualquier método que la proporcione al final de los periodos. Tiene el inconveniente de que, si la pendiente de la corriente es grande, el valor obtenido puede ser muy sensible a errores de sincronización

Este método se ha denominado "**MDesliz1y2_1**" y se ha implementado en un bloque C cuyo código se lista en el apéndice C, que se comenta en un punto posterior.

2.10.4 Muestreadores a fin de periodo de 2 a 32 muestras

Toman un determinado número de muestras (2, 4, 8, 16 y 32) distribuidas uniformemente en el tiempo de un periodo de modulación y entregan al final del periodo la media de las muestras tomadas.

En la siguiente figura se ilustra para un ejemplo de 8 muestras:

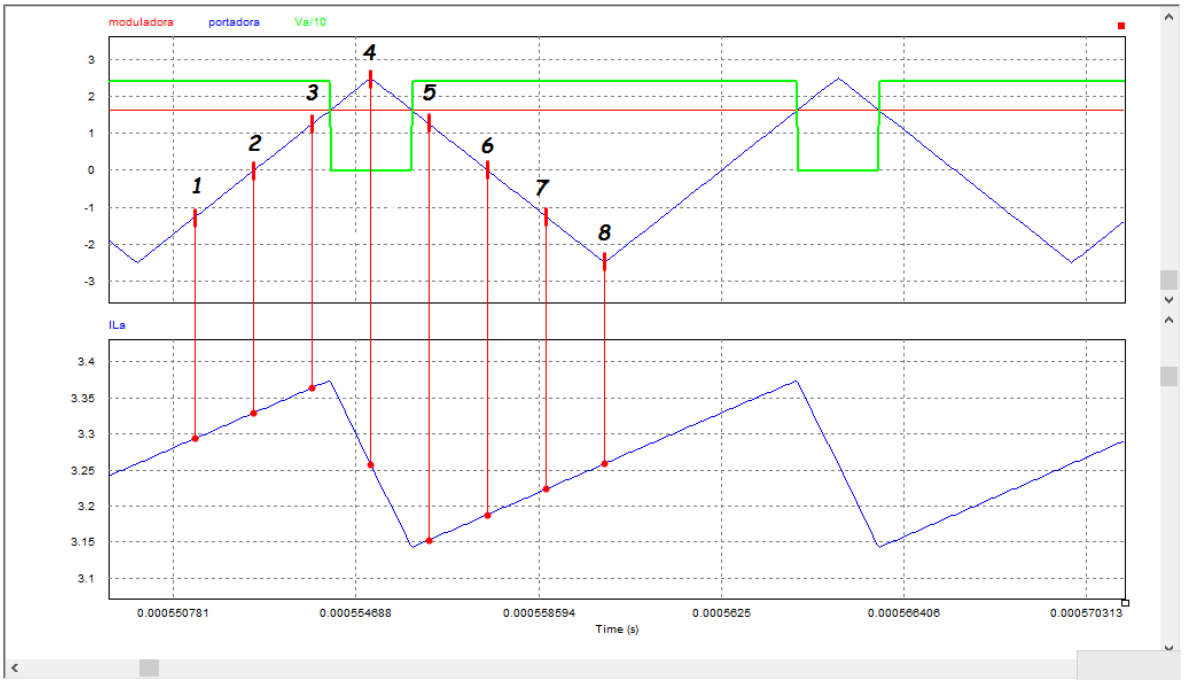


Fig. 2.13 Muestreo a fin de periodo de 8 muestras

Este muestreador toma 8 muestras en los instantes indicados (1 a 8) del periodo de modulación, halla su media y la entrega sólo al final de dicho periodo. Mientras, mantiene el valor entregado al fin del periodo anterior. Al aplicarle el SOH, éste suaviza las transiciones entre los puntos sucesivos.

Como se ve en la figura, la simetría hace que las parejas de muestras 1-7, 2-6 y 3-5 se compensen y den valores próximos al valor central. Las muestras 8 y 4 darían cada una de ellas, teóricamente, también el valor central.

La intención al aumentar el número de muestras es diluir el posible error de alguna de ellas, así como cubrir lo más posible todo el periodo, de modo que su media se aproxime cada vez más a la de la corriente media en él.

Tiene también la ventaja de que, al muestrear en varias zonas de las rampas de corriente, de pendientes diferentes, reduce mucho la sensibilidad a errores de sincronización, ya que la variación en unas rampas compensa la variación en otras.

Tiene a priori dos inconvenientes: que cuantas más muestras por periodo se tomen, se requeriría de un hardware más rápido; y que hasta el final de cada periodo no está disponible un nuevo valor.

Este método se ha denominado, según el número de muestras promediadas, "**MFinPer2**" a "**MFinPer32**".

Se ha implementado en un bloque C cuyo código se lista en el apéndice D.

Para no cargar la simulación, se aprovecha el mismo código para los 5 números de muestras diferentes, entregando la salida de cada uno de ellos por diferente patilla de la cajita del bloque.

El código sigue el mismo procedimiento explicado previamente para identificar los instantes de simulación en que debe tomar las muestras. Va acumulando sus valores en el array de acumuladores de tipo flotante "double" **`sAcum32_da[i]`** y, cuando identifica un fin de periodo de modulación, calcula las medias y las entrega a las patillas de salida out[i] correspondientes.

2.10.5 Muestreador deslizante de las últimas 2 a 32 muestras

Tiene los mismos propósitos que el muestreador del punto anterior, pero persigue evitar su inconveniente de tener que esperar al fin del periodo para actualizar su valor de salida.

Su esquema de funcionamiento se ilustra para un ejemplo de 8 muestras en la figura siguiente:

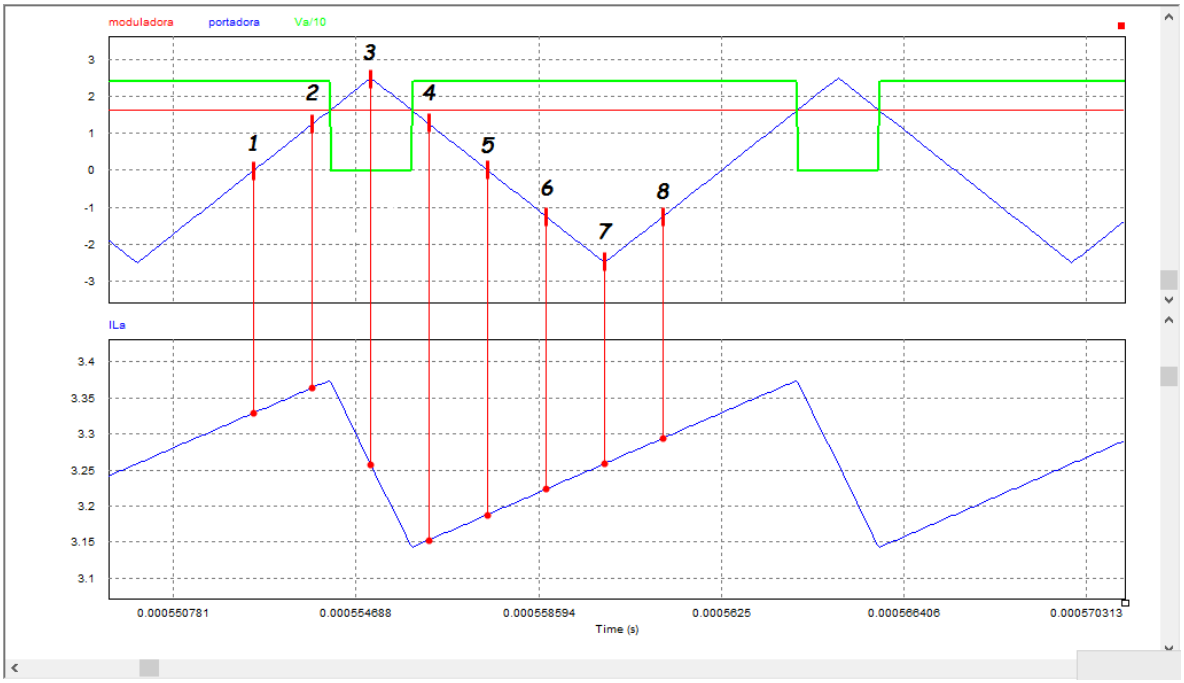


Fig. 2.14 Muestreo deslizante de las últimas 8 muestras

Se reparten 8 instantes de muestreo uniformemente en un periodo de modulación y, cuando llega el instante siguiente (en la figura, el instante número 8), se toma una nueva muestra, número 8, se halla la media de las últimas 8 más recientes (1 a 8), que se entrega en ese mismo instante a la patilla de salida. De este modo, un cambio de moduladora que ocurra en el interior de un periodo de modulación, cuando se produce la conmutación, cambia de posición en el tiempo la rampa correspondiente y, con ella, el valor medio de la corriente. Al tomar la primera muestra afectada por el cambio de rampa, esta muestra contribuye a la media con la octava parte del nuevo valor y así las sucesivas muestras tomadas de las nuevas rampas, hasta completar las 8. Esto supone una actualización progresiva de la nueva corriente media sobre el valor entregado por el muestreador, pero que comienza lo más próximamente posible al instante en que la modulación ha cambiado.

Para que la compensación entre muestras simétricas funcione correctamente, igual que en la figura 2.13 del método anterior la muestra 7 se compensaba con la 1, en la figura 2.14 de éste, la muestra 8 tiene que compensarse con la muestra 6, ya de la rampa más reciente del periodo de modulación anterior.

El método tiene, por tanto, los mismos ventajas e inconvenientes que el anterior del muestreador a fin de periodo, con las mejoras, a priori, de una actualización más próxima en el tiempo a la modulación y más progresiva.

Este método se ha denominado, según el número de muestras promediadas, "**MDesliz2**" a "**MDesliz32**".

Se ha implementado en un bloque C cuyo código se lista en el apéndice E.

Es parecido y está basado en el del anterior muestreador.

Para no cargar la simulación, se aprovecha el mismo código para los 5 números de muestras diferentes, entregando la salida de cada uno de ellos por diferente patilla de la cajita del bloque.

El código sigue el mismo procedimiento explicado previamente para identificar los instantes de simulación en que debe tomar las muestras.

Lo particular a reseñar de este código es que, para disponer de las últimas N muestras, las va guardando en un array de dobles que trata circularmente (cuando llega a un extremo, pasa al otro extremo), para ahorrar copias.

2.10.6 Muestreador deslizante de 1 o 2 muestras, actualizadas en varios instantes

Este método persigue llevar algo más lejos la idea de actualizar lo antes posible la posible variación en la corriente media respecto al momento de la variación de la modulación que la produce. Por seguir con el ejemplo de 8 muestras y atendiendo a la figura 2.13, si se produjera el cambio de moduladora entre los instantes 7 y 8, la muestra 8 ya lo recogería, pero, si tiene que repartir su contribución entre 8 muestras, su magnitud queda demasiado diluida, resulta una progresividad demasiado lenta: el efecto sobre la medida no sería completo hasta que la muestra más antigua (1) del tren llegara al instante de tiempo 8, es decir, todo un periodo de modulación. Se pretende abreviar la aparición en la salida del

muestreador del efecto completo de la modulación reduciendo a dos o a una el número de muestras que promediar. En el ejemplo, si en el periodo se distribuyen 8 instantes de muestreo; cuando se muestrea el 4, se entrega como medida su valor único (1 muestra); cuando se muestrea el 5, se entrega su media con la de su simétrico el 3 (2 muestras). Por tanto, en esta media aparece ya adelantado su efecto dividido sólo entre 2, en lugar de entre 8. Continuando, cuando se muestrea el 6, con la del 2; la del 7 con la del 1; y la del 8 ella sola. Con ello, en las muestras de los vértices, aparece ya la nueva media actualizada completa (es decir, como máximo, medio periodo de modulación más tarde) y, en las muestras intermedias, aparece adelantado la mitad de su efecto, como máximo 1 octavo de periodo de modulación más tarde.

El inconveniente puede ser que ese retardo pequeño reducido es variable: depende de en qué instante dentro de la rampa de la portadora aparezca la variación de la corriente y que tiene el mismo máximo que el muestreador en los vértices del punto 2.10.2, por lo que el controlador tendrá que estar diseñado para trabajar bien en el caso peor, sin poder aprovechar el adelanto que se consigue a veces.

Tiene también, como los muestreadores de pocas muestras, el inconveniente de su sensibilidad al error de sincronización.

Este método se ha denominado, según el número de instantes muestreados por periodo de modulación, "**MDesliz1y2_4**" a "**MDesliz1y2_32**". Puede observarse que los denominados (en los puntos 2.10.2 y 2.10.3) "**MDesliz1y2_2**" y "**MDesliz1y2_1**" pueden considerarse casos particulares de este tipo, donde, por distribuir sólo 2 o 1 instantes de muestreo dentro del periodo de modulación y caer, por tanto, en los vértices, la medida se genera a partir de una sola muestra. Por esto, todos ellos se han implementado en el mismo bloque C.

Su código se lista en el apéndice C. Se parece al del apéndice E de muestreado deslizante de las últimas 2 a 32 muestras, ya que debe ir almacenando las muestras obtenidas para poder hacer la media con las simétricas antiguas.

Cabe reseñar como particularidad el modo de generación del índice de la muestra simétrica a la actual:

```
j = - sam_ia[i];
while( j < 0 ) {
       j +=  ns_i;
       }

out[i] =  ( in[0] + sSam_daa[ i ][ j  ])  / 2.0;
```

El índice de la actual está en **`sam_ia[i]`**. Al sumar a su opuesto el número de instantes por periodo, que está en **`ns_i`**, se obtiene en **`j`** el índice de la simétrica y, con él, se obtiene el valor de dicha muestra, que está en el elemento **`sSam_daa[ i ][ j  ]`** de la matriz de dobles, de dimensión doble **`sSam_daa`**.

2.10.7 Second Order Hold SOH para suavizar los sensadores síncronos

La mayoría de sensadores síncronos presentados en los puntos anteriores entregan un valor fijo durante cada periodo de modulación Tm o fracción de él y dan una transición brusca al nuevo valor en el siguiente instante de actualización de la medida. Como ya se ha comentado en el punto 2.10.1 para el integrador reiniciado, para un controlador analógico conviene suavizar dichas transiciones, lo que se va a hacer con el “Second Order Hold” sobre todos estos sensadores, salvo para alguno que se vea que no conviene.

2.12 Criterios de valoración de los métodos de sensado

Para comparar los diversos métodos de sensado estudiados, conviene establecer unos criterios de valoración de los mismos que permitan calificar su utilidad para el propósito para el que se quieren usar, que es para facilitar un control de calidad de la corriente de salida.

Para dicho propósito, lo que interesa es que los métodos de sensado den una medida lo más exacta posible del módulo de la corriente de salida y que la den añadiendo el mínimo retardo posible al proceso, de modo que permitan una corrección rápida de las posibles perturbaciones por parte del controlador.

Para asegurar su validez para el inversor trifásico, que se ha definido como requisito, la medida se obtendrá a partir de unas sinusoides generadas por el inversor, que estarán “ensuciadas” por un rizado bastante fuerte procedente del proceso de modulación PWM.

Una acción importante del sensado de cara al control consistirá en reducir lo máximo dicho rizado.

Por tanto, en resumen, se valora:

La exactitud en el módulo de la medida.

Un tiempo de retardo lo menor posible.

El mínimo rizado residual.

Para medir los posibles errores de módulo, es necesario contar con un patrón o referencia con el que comparar. Como se ha comentado anteriormente, existe un método de sensado que da la medida exacta que se quiere obtener: la media de la corriente entregada al filtro en un periodo de modulación. Éste es el **integrador analógico reiniciado** en los periodos de modulación, que, por tanto, **se usará como referencia**.

Un indicador de la exactitud del módulo es el **error de amplitud del armónico fundamental** de 50 Hz de la potencia generada, dado que es ese armónico fundamental la magnitud que cabe esperar que un generador de alterna trifásica de calidad entregue correctamente. Este indicador puede medirse mediante la utilidad “FFT” del Simview sobre la curva de medida entregada por el sensador a lo largo de una sinusoidal.

Otro indicador muy interesante es la **distorsión armónica total**, que mide tanto la deformación de la señal de salida como la presencia de armónicos de alta frecuencia procedentes de la modulación. Es otra magnitud que también permite obtener el Simview en postproceso, con su utilidad “THD”.

Otro indicador muy importante de la exactitud de la medición es la raíz del **error cuadrático medio (Root Mean Square, RMS)** entre la magnitud entregada por los métodos de sensado y la referencia. Para obtenerlo, a falta de un elemento en la biblioteca del PSIM, se ha desarrollado otro bloque en C (que se lista en el apéndice F) que lo calcula en los instantes el tiempo de simulación y que se aplicará a un ciclo completo de 50 Hz como representativo de las condiciones de trabajo esperadas.

Un indicador del retardo añadido por el método de sensado, muy interesante desde el punto de vista del control del inversor, es el **desfase en función de la frecuencia**, dado por la

gráfica de fase del **diagrama de Bode**, que permite obtener el PSIM realizando un barrido en frecuencia con su utilidad “AC Sweep” y que puede procesarse posteriormente en el Simview para extraer de él medidas a frecuencias representativas.

Y, por último, otro indicador muy directo del retardo añadido por cada método de sensado es el **tiempo de respuesta** de la medida de corriente media **a un escalón** de una señal de entrada al sistema, ya sea de señal de control “ma” o de perturbación. Por analogía con la caracterización de los componentes electrónicos, se ha definido como el tiempo entre el escalón de excitación y el instante en el que la respuesta entra y permanece en una banda del 10% alrededor de su nuevo valor permanente. Un modo de obtenerlo es en postproceso en Simview, añadiendo las rectas horizontales que definen la banda del 10% y midiendo con los cursores el diferencial de tiempo desde el salto de la entrada y el tiempo cuando la señal entra en dicha banda.

En la figura siguiente se ilustra la medición de ese tiempo de respuesta para la medida entregada por el sensador LPFSimple10k (curva verde):

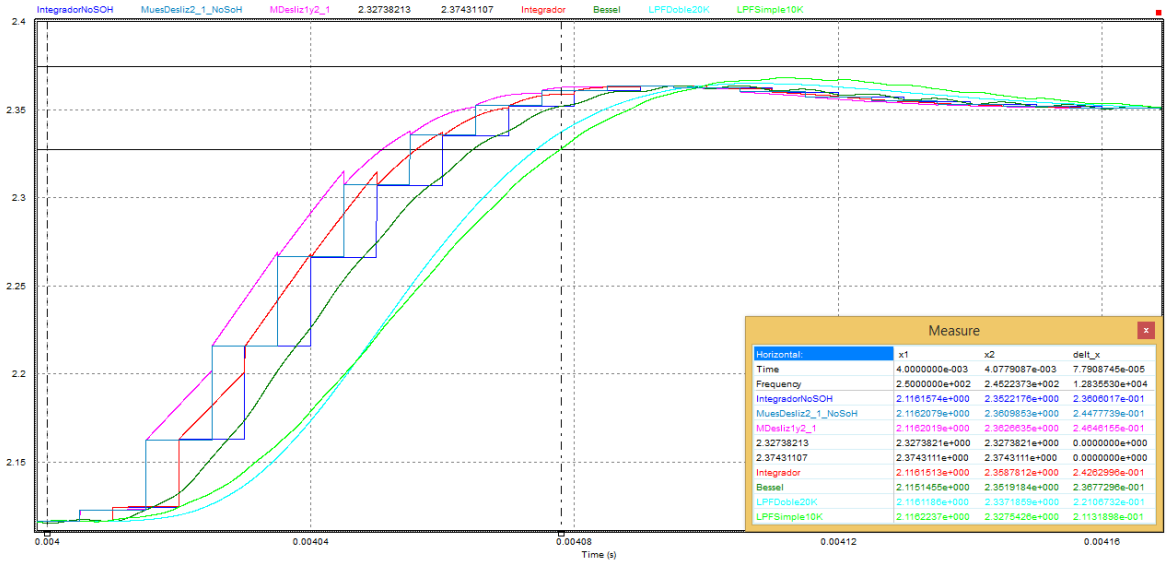


Fig. 2.15 Medida del tiempo de respuesta de varios métodos de sensado

En esta figura, conviene también observar un detalle muy importante, para dar el valor correcto a los resultados: La curva rosa (del sensador Mdesliz1y2_1, adelanta por todo un periodo de modulación a la azul oscuro del sensador “IntegradorNoSOH” (los escalones tienen una anchura horizontal de 10 µs, que es el periodo de modulación). Es una diferencia

muy importante: significa que la rosa está prediciendo al principio del periodo de modulación la media de la corriente en todo el periodo. Sin embargo, su diferencia relativa queda dividida entre los 60 µs de los 6 escalones que tardan todas las curvas en subir. Estos 60 µs no se deben propiamente al método de sensado, sino a la respuesta del circuito (concretamente, al filtro LC). Por tanto, en este caso, es importante prestar atención a las diferencias absolutas entre los tiempos de respuesta y no a las relativas.

En la tabla siguiente se resumen los anteriores indicadores y sus métodos de obtención:

Indicador	Método de obtención
Error de amplitud del primer armónico	FFT con PSIM de un ciclo de 20 ms
Deformación y contenido de armónicos superiores	THD de Simview de un ciclo de 20 ms (pero dan error "-1.#IND000e+000")
Error cuadrático medio RMS	Bloque medidor en C Resultado al final de un ciclo de 20 ms
Desfase	Tabla de desfases a 1KHz, 10 KHz y 40 KHz extraída del Bode.
Tiempo de establecimiento de escalón de entrada	Medidas en SimView sobre respuesta a escalón con líneas de banda de 10%

Tabla 2.1 Indicadores de valoración de los métodos de sensado

3. DISEÑO DEL INVERSOR VSI Y SIMULACIÓN EN PSIM

3.1 Introducción

En el presente capítulo se va a realizar, previamente al estudio de los métodos de sensado de la corriente, el diseño de un inversor trifásico sobre el que probarlos. Dado que el presente trabajo TFG tiene como requisito que los métodos de sensado sean válidos para un inversor trifásico y que, por tanto, se requiere la comprobación en él del funcionamiento del método o métodos de sensado que se propongan como solución, se ha decidido economizar los diseños aprovechando para toda la investigación el inversor trifásico implementado para dicha comprobación.

Para este diseño, se ha decidido utilizar PSIM, una potente herramienta software de diseño y simulación de circuitos electrónicos, especializada en circuitos de potencia. Esto ha permitido el estudio detallado de este tipo de circuitos, pudiendo obtener todo tipo de medidas de las magnitudes eléctricas de los circuitos, con mucha mayor seguridad para el personal, facilidad y economía que en implementaciones reales, tanto de material, como, aún más importante, de tiempo de desarrollo. Gracias al enfoque realista de esta herramienta, el posterior salto de la simulación a la ejecución física real se espera que presente las menores diferencias y dificultades posibles.

Sobre el inversor trifásico diseñado en este capítulo, se irán aplicando, en el siguiente, los diversos métodos de sensado de la corriente de salida, se realizará una simulación de funcionamiento en lazo abierto (es decir, sin todavía efectuar el control de la magnitud de salida) y se tomarán medidas que caractericen su comportamiento, orientadas a una comparativa entre dichos métodos de sensado.

3.2 Especificaciones de diseño del inversor VSI

Con el mismo o parecido esquema de principio, existe gran diversidad de inversores, para diferentes campos de aplicación y, por tanto, con diferentes tensiones de entrada, potencia, métodos de control, etc.

Para el presente trabajo, se ha preferido especificar y simular un inversor trifásico de baja tensión y baja potencia, de modo que, caso conveniente, se facilitaría el posible salto futuro a una implementación física real en laboratorio, donde suele haber menor disponibilidad de componentes de alta tensión y, sobre todo, ofrecerían mucha mayor seguridad para el personal que los manipula.

Por ello, se han establecido las siguientes especificaciones para este inversor trifásico:

1.- Generador de tensión alterna trifásica equilibrada

2.- Frecuencia de salida:	50 Hz
3.- Potencia activa de salida:	30 W
4.- Tensión de pico de cada fase:	9 V
5.- Fuente de tensión de entrada continua única de:	24 V
6.- Portadora PWM triangular de frecuencia:	100 KHz

La frecuencia de la portadora se ha especificado bastante alta, para facilitar un adecuado filtrado a la salida del rizado debido a dicha portadora, que no requiera componentes excesivamente voluminosos y caros.

Como contrapartida, una frecuencia alta supone unos tiempos menores que presentan mayor exigencia a las velocidades del procesador o elemento de control, así como a los sensores, con lo que van a ponerse efectivamente a prueba los sensores que formen parte de los métodos de sensado bajo estudio.

3.3 Diseño del inversor trifásico VSI

Se va a utilizar la topología y esquema presentados en la figura siguiente:

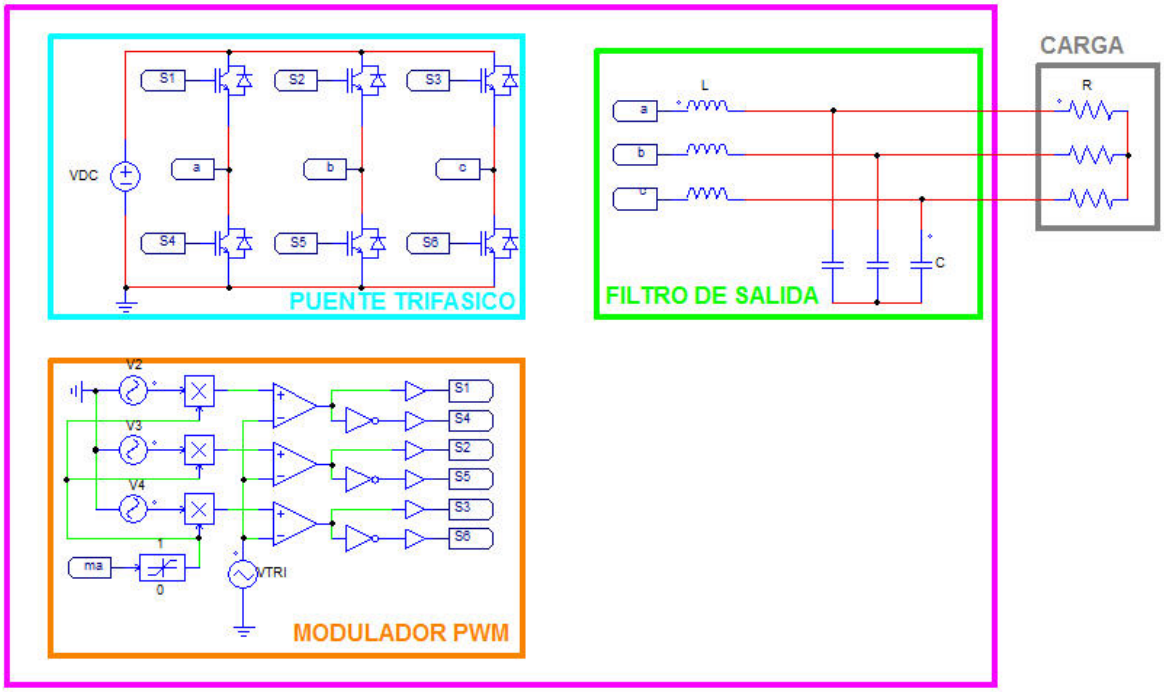


Fig. 3.1 Esquema del inversor trifásico VSI con su carga

Como se aprecia, se compone de los siguientes bloques:

1.- Modulador PWM

2.- Bloque de potencia, con el puente trifásico

3.- Filtro LC de salida

4.- Carga (aunque estrictamente no forma parte del inversor, es necesario incluirla para realizar las simulaciones)

Para los efectos del presente estudio, no se ha considerado necesario modelar las características reales de los componentes ni los tiempos muertos en el modulador PWM.

Se explica a continuación el diseño de cada uno de los bloques anteriores:

3.3.1 Para el modulador PWM se utiliza un generador de portadora triangular centrada en 0V y que varía entre -2.5 V y 2.5 V, que se aplica a los comparadores. En las otras entradas de éstos, se aplican las tres tensiones sinusoidales desfasadas 120º, también centradas en 0V y con tensiones de pico de 2.5 V, pero previa multiplicación por una tensión "ma", índice de modulación, limitado entre 0 y 1V, que permitirá la regulación más adelante, cuando se cierre el lazo. Para las especificaciones de diseño, este "ma" deberá tener un valor nominal:

$$ma = \frac{\frac{V_{DC}}{2}}{V_{pf}} = 0 \tag{3.3.1}$$

Este valor proporcionará una eficiencia suficiente, a la vez que deja un margen bastante amplio para la regulación frente a las perturbaciones cuando se cierre el lazo.

3.3.2 Para el bloque de potencia, se aprovecha también otra ventaja que ofrece la herramienta PSIM: que ahorra la complejidad de los drivers reales de los transistores conmutadores mediante el uso de unos drivers simbólicos.

3.3.3 La carga se supone en principio resistiva pura, aunque más adelante se estudiará también el efecto de una posible componente inductiva. La resistencia de carga de cada fase, Rc, deberá valer, para cumplir la especificación de potencia activa:

$$Rc = \frac{{V_{ef}}^2}{\frac{P}{3}} = \frac{{\frac{V_{pf}}{\sqrt{2}}}^2}{\frac{P}{3}} = 4.05\ \Omega \tag{3.3.2}$$

3.3.4 La capacidad C del filtro LC de salida se dimensiona para que presente a la frecuencia de modulación f_M una impedancia la décima parte de la resistencia de carga nominal, de modo que la eficacia del filtrado no se viera reducida cuando se conectara una carga de mayor resistencia que la nominal.

$$\frac{1}{2\pi * f_m * C} = \frac{Rc}{10} \tag{3.3.3}$$

$$C = \frac{10}{2} * \pi * Rc = \frac{10}{2} * \pi * 4.05 = 3.93\ \mu F \tag{3.3.4}$$

Se redondea a 4 µF

Para dimensionar la inductancia L, se tiene en cuenta que forma con la anterior capacidad un filtro LC, por tanto, de orden 2 y caída de 40 dB/ década desde la frecuencia de cruce. Se busca una frecuencia de cruce f_C que asegure una atenuación fuerte a la frecuencia de modulación de 100 KHz. Esta atenuación sería mayor cuanto menor fuera dicha fc. Sin embargo, dicha fc va a ser la que limite el ancho de banda del sistema en lazo abierto y, por tanto, la velocidad de respuesta posible para su control en lazo cerrado. Por ejemplo, si se fijara fc hacia los 1KHz, produciría una atenuación muy grande (80dB) a la frecuencia de modulación, pero los tiempos de respuesta de la acción de control frente a perturbaciones deberían ser bastante largos. Como solución de compromiso, se elige una frecuencia de cruce de 10 KHz, que producirá una atenuación muy apreciable (40dB) a la frecuencia de modulación y permitirá un control rápido.

Según lo anterior, a continuación se calcula la L para una frecuencia de cruce de f_C de 10 KHz:

$$f_c = \frac{1}{2\pi * \sqrt{LC}} \tag{3.3.5}$$

$$L = \frac{1}{(2\pi * f_c)^2 * C} = 64\,\mu H \tag{3.3.6}$$

Para facilidad de encontrar un componente comercial, se redondea hacia arriba a 100 µH

3.4 Simulación con el inversor trifásico VSI en lazo abierto

Para observar y validar el anterior diseño y comprender mejor el problema a resolver con el método de sensado, se va a realizar una simulación con el inversor trifásico recién diseñado, en lazo abierto, y se van a tomar gráficos de sus principales magnitudes eléctricas: por un lado, las tensiones y corrientes en la carga; por otro, para estudiar cómo realizar el sensado, las tensiones y corrientes en las inductancias del filtro de salida.

Para esto, se añade al esquema en PSIM los medidores de tensión y corriente oportunos, en los puntos que muestra la siguiente figura:

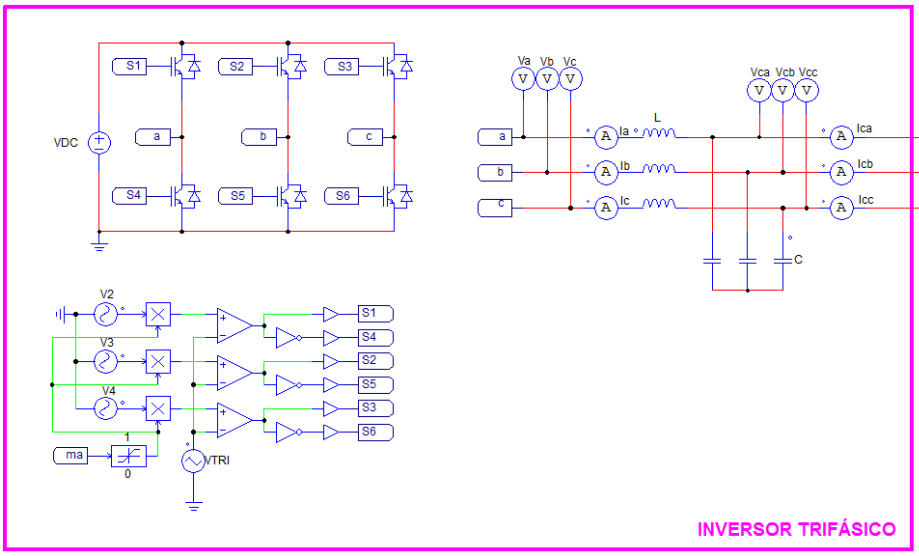


Fig. 3.2 Inversor con voltímetros y amperímetros para obtención de magnitudes eléctricas de interés

Para efectuar la simulación, hay que programar ciertos parámetros en PSIM, lo que se realiza dentro del elemento “Simulation Control”, dentro de la siguiente ventana de parámetros:

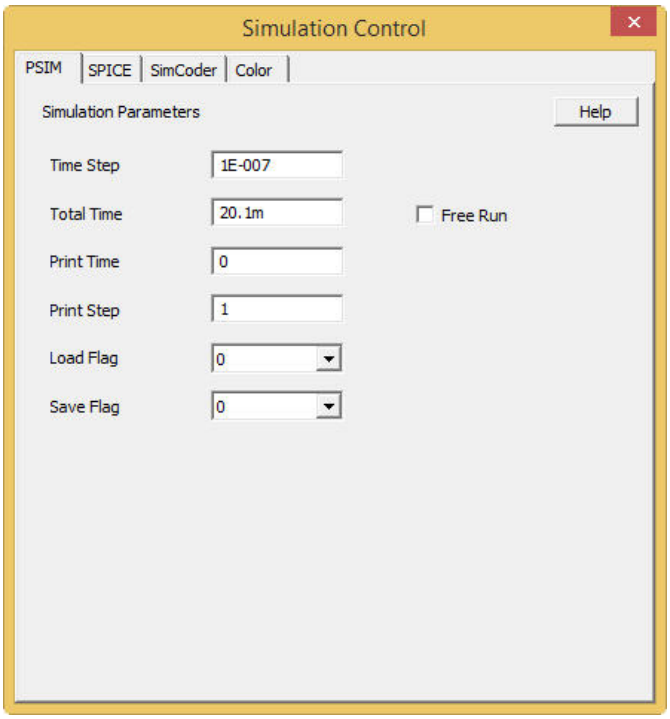


Fig. 3.3 Esquema del inversor trifásico VSI con su carga

El principal es el “Time Step”, que define el paso de tiempo de simulación, en segundos. Es importante ajustarlo bien para trabajar con suficiente precisión, pero que las simulaciones tarden lo justo. Para una simulación rápida, interesa un valor lo máximo posible, pero eso puede hacer perder precisión. Se recomienda (y, en ocasiones, PSIM lo hace automáticamente) ajustar un valor 10 veces menor que el mínimo tiempo significativo del circuito y sus señales.

El “Total time” define el tiempo total de simulación y también interesa ajustarlo correctamente para simular un tiempo suficiente para observar el comportamiento del sistema, pero no excesivo, para no alargar la ejecución de las simulaciones.

Los resultados de la simulación, PSIM los escribe a un fichero que pasa a su módulo visualizador SIMVIEW, que ofrece bastante flexibilidad para configurar la visualización, así como añadir algunas funciones matemáticas interesantes sobre los datos visualizados (como FFT, THD, etc.).

Una vez ajustados los parámetros, se lanza la simulación, con lo que se obtienen las siguientes curvas de evolución de las magnitudes medidas en la carga, presentados por SIMVIEW:

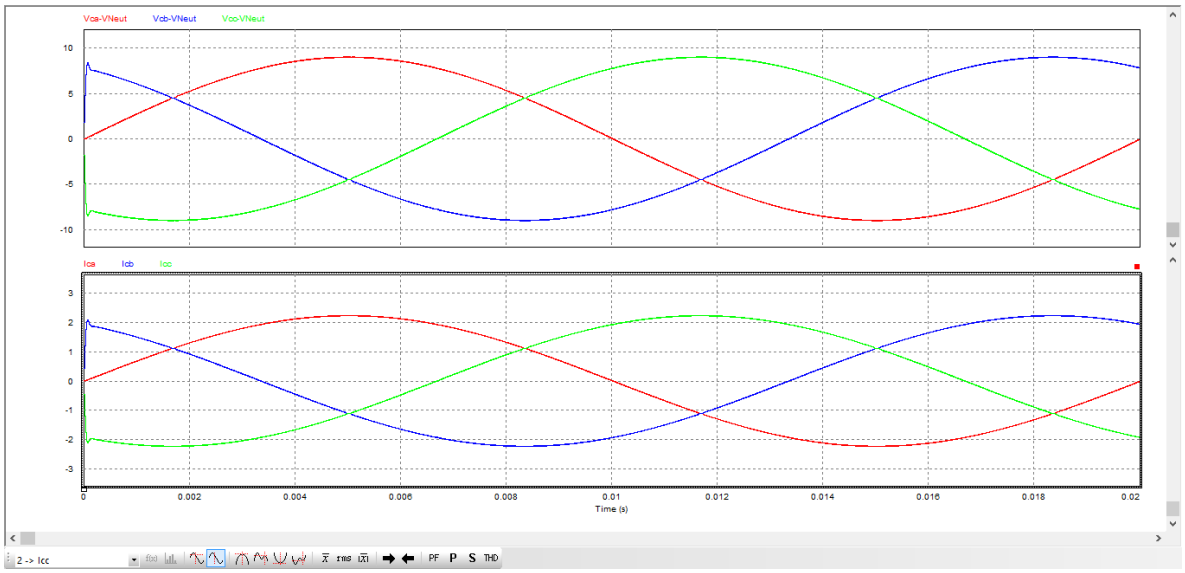


Fig. 3.4 Tensiones y corrientes de las 3 fases sobre la carga

Puede observarse cómo, en efecto, la tensión de pico de la tensión es de 9 V y la corriente produce una potencia de 10 W sobre cada fase, cumpliendo las especificaciones.

A continuación, en el siguiente gráfico, se presentan las corrientes por las inductancias del filtro de salida producidas por el proceso de la modulación, que son fundamentales para estudiar los métodos de sensado, así como las magnitudes principales de dicha modulación:

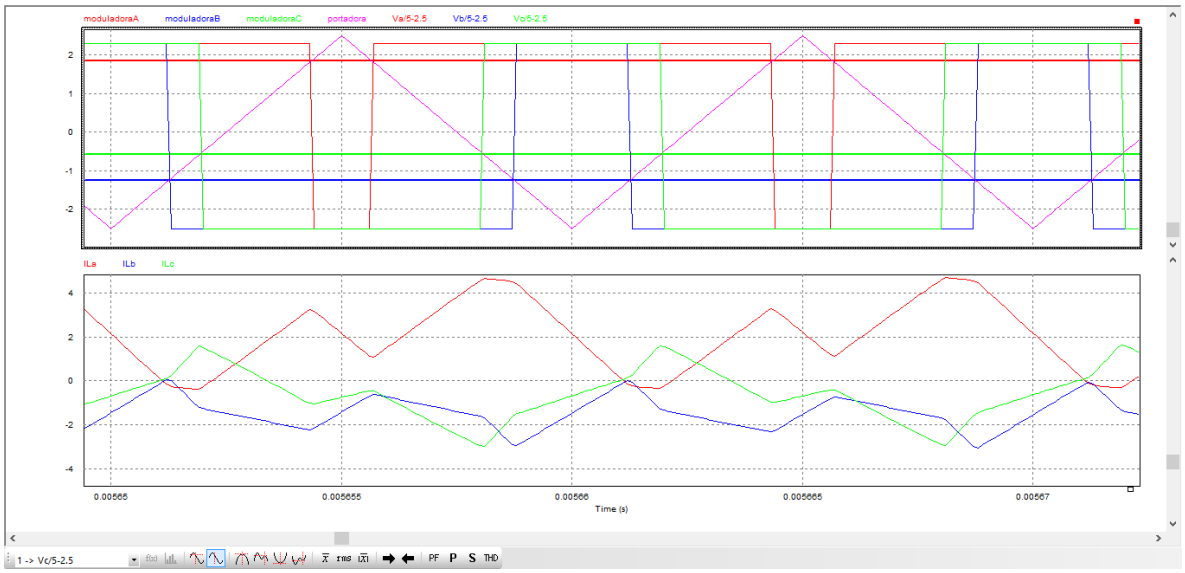


Fig. 3.5 Modulación PWM de las tres fases y corrientes producidas sobre la inductancia.

Es muy importante entender lo mejor posible este proceso y la evolución de las corrientes por la inductancia del filtro, para pensar y estudiar los diferentes métodos de sensado de dichas corrientes, que es el objeto del presente trabajo. Por ello, en el punto siguiente se va a intentar profundizar en su comportamiento.

3.5 Detalle de la evolución de la corriente por la inductancia del filtro de salida

En la anterior figura 3.5 puede observarse cómo la corriente de cada una de las fases cambia en rampas rectas entre los instantes de conmutación de cualquiera de las tres fases (se produce una nueva rampa en cada conmutación). Esto demuestra una interacción entre las tres fases que complica el análisis (interacción que podría considerarse que ocurre a través de la tensión del neutro).

Sin embargo, el principio de superposición puede facilitar la comprensión del proceso: la corriente total a través de la inductancia de una fase es la suma de las corrientes que procederían de cada una de las 3 fases cuando las demás fueran nulas. En esa superposición sobre, por ejemplo, la fase A, la corriente propia de A se suma con coeficiente 1 y la

corriente de cada una de las otras, B y C, se resta (porque circula en sentido opuesto) con coeficiente 0.5 (porque, por ejemplo, la B, se reparte entre la A y la C).

Es importante entender cómo la corriente propia de cada fase, con las otras 2 anuladas, dentro de un periodo de modulación (subida y bajada de la portadora triangular) es una rampa recta entre los dos extremos del periodo, correspondiente a la respuesta a la tensión sobre la inductancia cuando los conmutadores todavía no han conmutado, interrumpida por otra rampa recta diferente, correspondiente a dicha respuesta cuando los conmutadores han conmutado. Estas conmutaciones de modulación de esta fase ocurren a valores de tensión de la portadora iguales (por coincidir con la tensión de su sinusoide moduladora, si se considera que no ha cambiado dentro de un periodo). Por tanto, dicha rampa queda centrada dentro del periodo de la portadora triangular, con su centro en el instante de tiempo situado en el pico de la portadora triangular.

En la figura 3.5, en la curva de ILa, en el primer periodo de la portadora, en el intervalo de tiempo comprendido entre los instantes de conmutación de la fase C (por no se producirse ya más conmutaciones de las fases B y C), puede observarse dichas dos rampas de la corriente ILa por las dos conmutaciones de A. Puede observarse también la simetría de la corriente ILa respecto al centro del periodo.

Por tanto, la corriente media de cada una de las dos rampas (de la componente de superposición de la fase A, con las B y C anuladas) en el periodo de la portadora coincide con el valor instantáneo de dicha corriente en el punto central de dicho periodo.

Dado que la misma coincidencia ocurre para las componentes de superposición de cada una de las tres fases y que la portadora es la misma para las tres, también ocurrirá con la superposición de estas tres componentes sobre cada una de las fases, es decir: el valor medio en un periodo de la portadora triangular de la corriente por la inductancia coincidirá con su valor instantáneo en el centro del periodo, es decir, en el instante temporal del pico del triángulo.

Análogamente podría explicarse que esa simetría ocurriría también respecto a los valles de la portadora triangular, si los valores de la moduladora no cambiaran entre los dos periodos. Normalmente, por la gran diferencia de frecuencias de la moduladora y la portadora, dichos valores cambian lentamente entre 2 periodos de modulación.

Resumiendo lo anterior, la expectativa de poder obtener una medida de la corriente media en un periodo de portadora muestreando en el centro del periodo depende de que las rampas de corriente sean rectas, lo que ocurre cuando la constante de tiempo del filtro es bastante mayor que el periodo de la portadora.

Vamos a analizar ahora el caso de que no sea así, sino que las transiciones de las corrientes por la inductancia sean curvas de carga. Eso ocurre, por ejemplo, cuando el filtro carece del condensador (o su capacidad es muy pequeña) y la inductancia L es pequeña. Se presenta ahora las mismas curvas con C = 1pF y L = 5 µH:

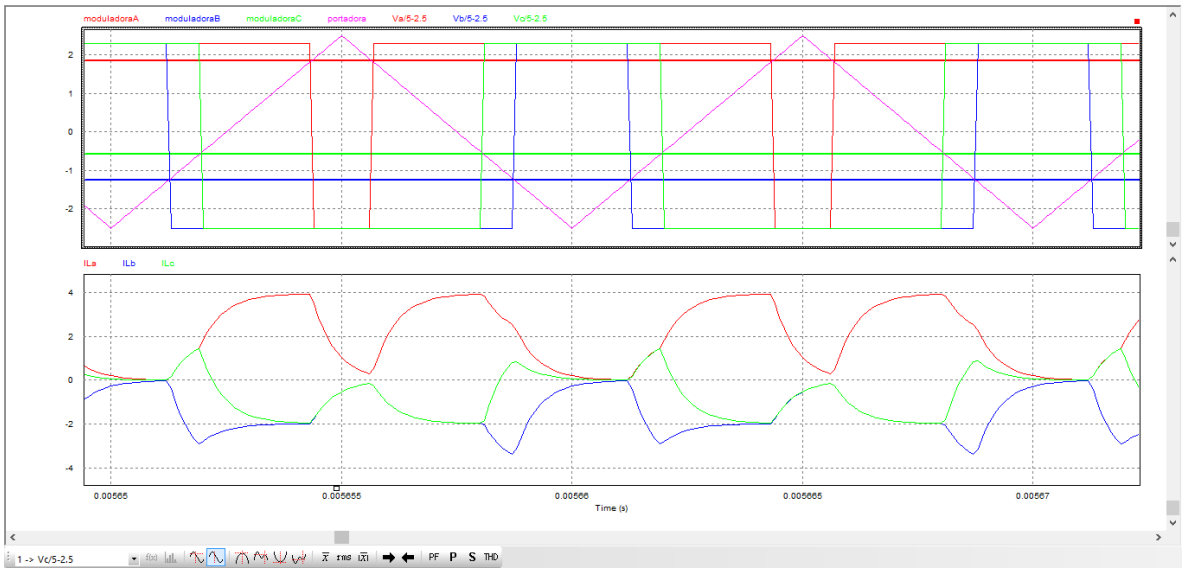


Fig. 3.6 Modulación PWM de las tres fases y corrientes producidas sobre la inductancia con constante de tiempo del filtro pequeña.

En este caso, claramente se aprecia que ya la corriente instantánea en el centro del periodo de portadora difiere mucho de la media de la corriente en el periodo, por lo que habrá que buscar otros métodos de sensado. Una posibilidad a estudiar (se hará en un capítulo posterior) es el “multisampling”, es decir, tomar varias muestras de la corriente instantánea por cada periodo, distribuidas uniformemente.

4. SIMULACIONES, RESULTADOS Y COMPARATIVA

4.1 Introducción

En el presente capítulo se van a presentar las simulaciones realizadas para valorar los diferentes métodos de sensado estudiados, los resultados de las mismas y unas comparativas entre ellos.

Tras la implementación en PSIM, documentada en los capítulos anteriores, del sistema de prueba y de los 23 diferentes métodos de sensado a comparar, de la definición de los indicadores de calidad a obtener para ellos y de los procedimientos para su obtención, se han llevado a cabo las simulaciones necesarias en PSIM del sistema trifásico en lazo abierto.

Primeramente, se ha realizado la simulación en tiempo, durante un tiempo de simulación de poco más de 20 ms, para cubrir un ciclo de la sinusoidal. En la figura siguiente se presenta la visualización en Simview del resultado de dicha simulación, en la que se incluyen las medidas de todos los sensores bajo estudio. Como se ve, todas las curvas aparecen superpuestas, muy juntas, debido a que los métodos estudiados dan medidas de módulo muy parecido.

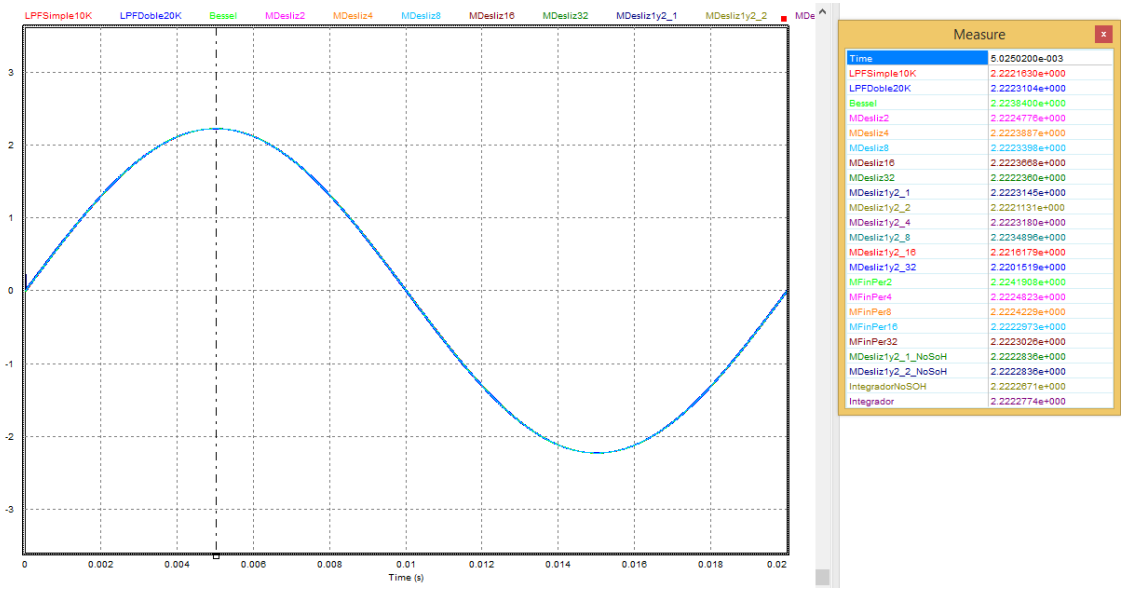


Fig. 4.1 Medidas entregadas por todos los métodos de sensado

De la similitud de las magnitudes medidas da idea la tabla de medidas obtenidas en Simview en el instante de la cresta de la sinusoide, a 5 ms, que resultan iguales en muchos decimales.

En la misma simulación, se ha utilizado la herramienta “FFT”, de Simview, para generar en postproceso el espectro de cada una de las curvas presentadas. Realizando un zoom alrededor de los 50 Hz del armónico fundamental y tomando las medidas con el cursor, se obtiene una tabla con los valores de dicho armónico fundamental de todas las curvas. En la siguiente figura se ilustra el proceso:

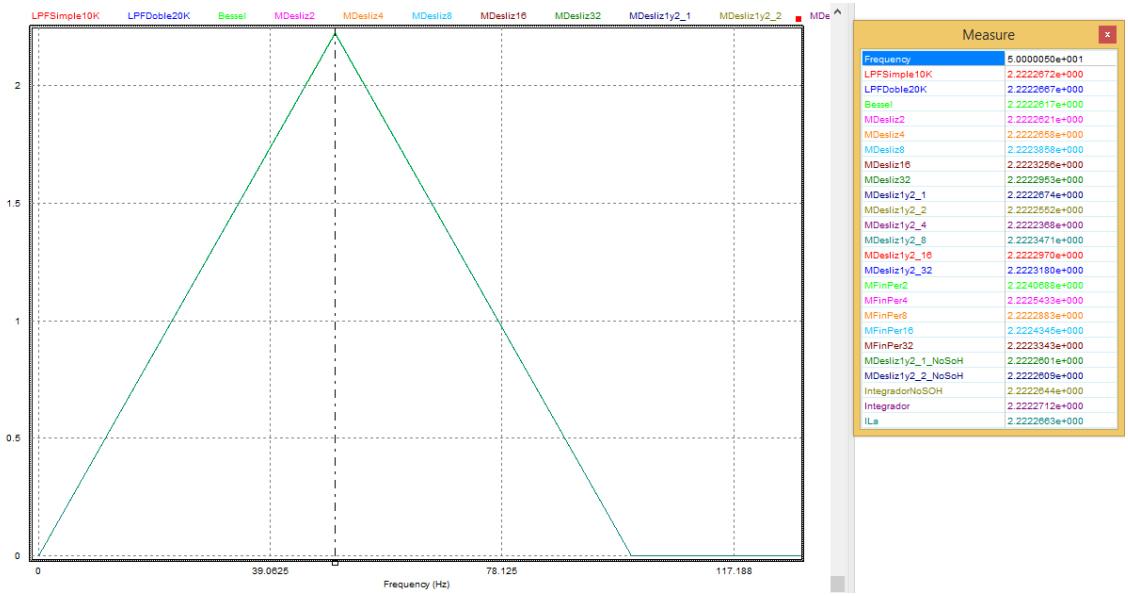


Fig. 4.2 Obtención del armónico fundamental de 50 Hz de todas las medidas entregadas por todos los métodos de sensado

También sobre dicha simulación se ha obtenido en postproceso con la herramienta "THD" del Simview la distorsión armónica total de cada curva, que luego se llevará a los resultados. Sin embargo, por algún tipo de fallo del Simview, para muchas de las curvas ha dado el valor de error "-1.#IND000e+000", como se muestra en la figura siguiente:

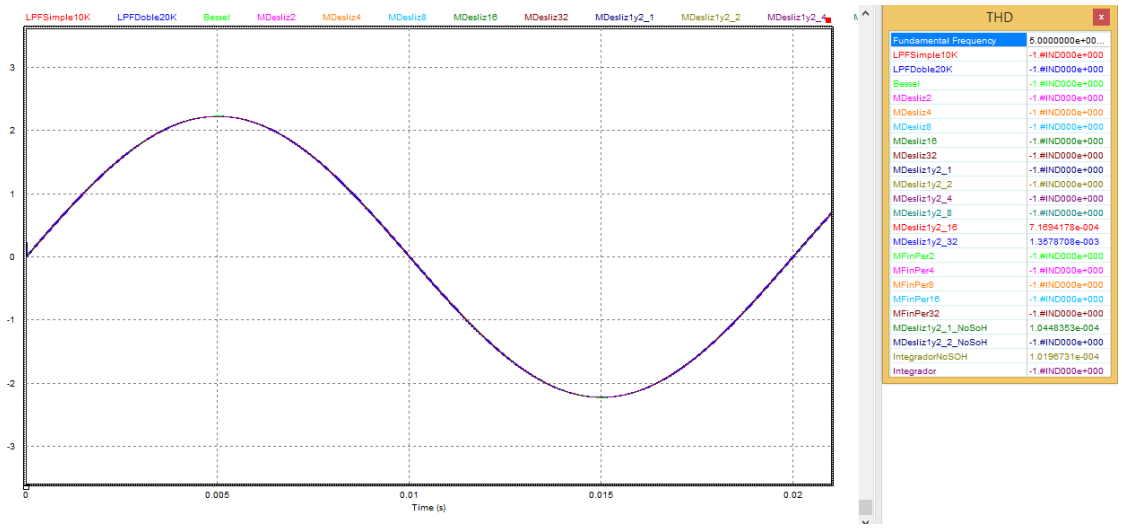


Fig. 4.3 Obtención del THD de las curvas de los sensadores

Se ha intentado corregir el error aplicando el medidor de THD a una sola de las curvas, pero sigue dándolo, como muestra la siguiente figura:

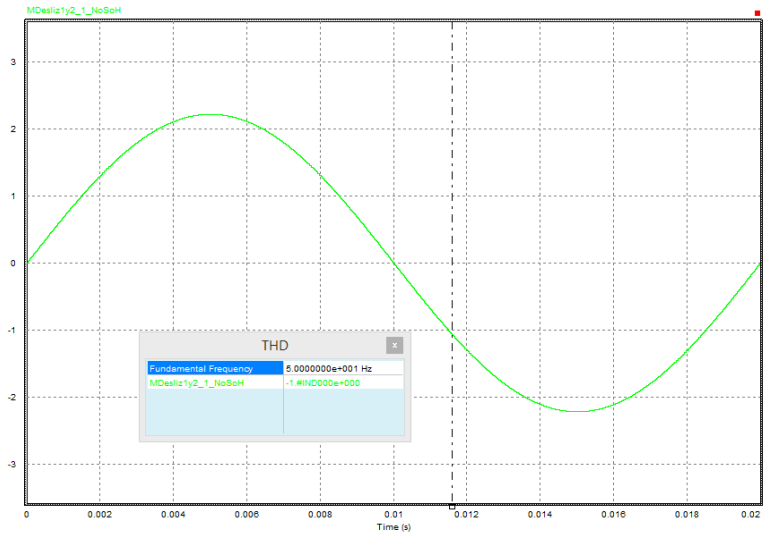


Fig. 4.4 THD de una curva con valor de error "-1.#IND000e+000"

Sin embargo, en alguna ocasión en que esporádicamente ha dado un valor sin error, han dado valores inferiores a 1e-3.

Continuando con la simulación de tiempo, se presenta en el visualizador de Simview las salidas de los elementos medidores de error RMS, implementados en bloque de C, que se aplicaron a todas las salidas para calcular su error respecto al patrón del integrador, con objeto de recoger el valor que cada uno ha calculado a lo largo de un ciclo de 20 ms. En la siguiente figura se ilustra el proceso:

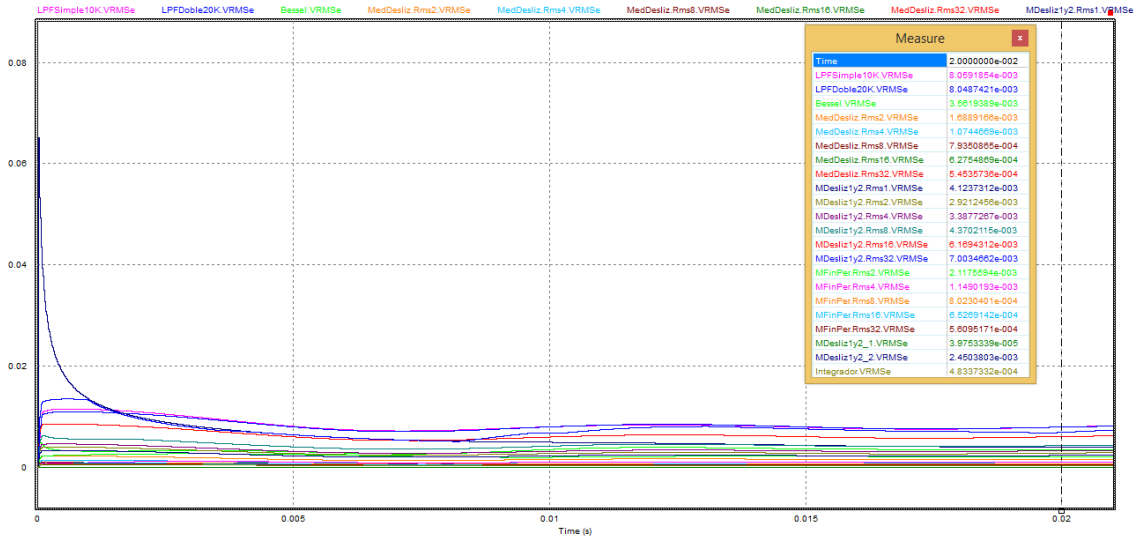


Fig. 4.5 Recogida de los errores RMS respecto a la referencia calculados por el bloque C

Los valores se recogen en una tabla colocando el cursor de medida en el tiempo 20 ms, del fin del primer ciclo de 50 Hz y se trasladan a los resultados.

Posteriormente, se ha realizado una simulación de barrido AC (AC Sweep) con PSIM, que ha generado unos diagramas de BODE que se presentan en la siguiente figura:

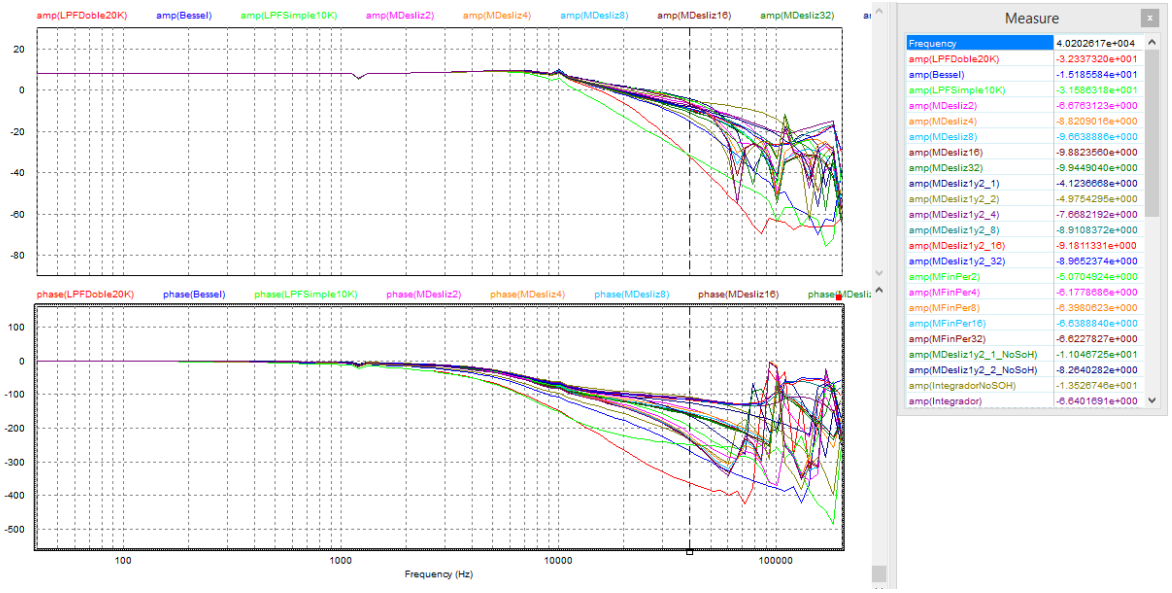


Fig. 4.6 Diagrama de Bode de las medidas de todos los sensadores y extracción de valores a 40 KHz

De dichos diagramas, se ha extraído los valores en los puntos de 50 Hz, 1 KHz, 10 KHz y 40 KHz, como se ilustra en la figura para el caso de 40 KHz.

Y, por último, se ha realizado una simulación en tiempo de unos 1 ms con las sinusoidales detenidas en la fase de 60º y con un salto de la R de carga en el instante 0.5 ms, cuyo resultado para un subconjunto de los sensadores se presenta en la siguiente figura:

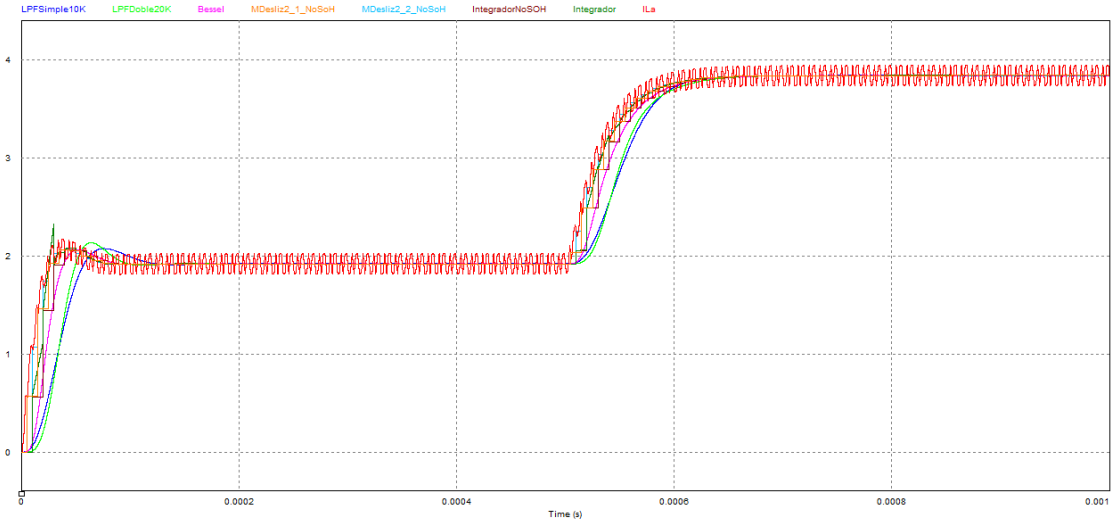


Fig. 4.7 Respuesta de algunos sensadores a un escalón de Resistencia de carga

El escalón inicial es de arranque del sistema desde las condiciones iniciales.

Se han ido representando sensadores, por subgrupos, para permitir la visualización, y tomando para ellos medidas de sus tiempos de respuesta, por el procedimiento que se ha explicado en el punto 2.12.

4.2 Resultados de las simulaciones. Valoraciones

Por los procedimientos definidos en el anterior punto 2.12, se han tomado los resultados y se han recogido, para su fácil comparación y presentación gráfica a la siguiente tabla de Excel:

			RMS	Bode 50 Hz		Bode 1KHz		Bode 10 KHz		Bode 40 KHz		Tiempo de	Amplitud
Método de sensado	FFT (50 Hz)	THD	(x1e-3)	mplitud (dB	Fase (º)	nplitud (d	Fase (º)	mplitud (dB	Fase (º)	Amplitud (dB)	Fase (º)	spuesta (u	de pico
LPFSimple10K	2,216626	-1.#IND00	8,059	8,05744	-0,4287	8,14837	-11,0310	5,44641	-150,1929	-31,45360	-248,6936	77,909	2,222138
LPFDoble20K	2,216625	-1.#IND00	8,049	8,05734	-0,4283	8,14795	-11,0062	7,92994	-146,2660	-32,13812	-363,1342	74,187	2,222186
Bessel	2,216621	-1.#IND00	3,562	8,05740	-0,2470	8,14302	-7,5666	7,97589	-106,0722	-15,05031	-267,8270	64,590	2,222138
MDesliz2	2,216621	-1.#IND00	1,689	8,06362	-0,0331	8,14611	-3,9738	8,57134	-70,5110	-6,62254	-142,7562	54,250	2,222210
MDesliz4	2,216625	-1.#IND00	1,074	8,05871	-0,0817	8,14428	-4,3846	8,32317	-74,1786	-8,75675	-147,1333	55,551	2,222223
MDesliz8	2,216745	-1.#IND00	0,794	8,04106	-0,0320	8,14240	-4,5832	8,23913	-76,1674	-9,59117	-153,8923	56,082	2,222163
MDesliz16	2,216685	-1.#IND00	0,628	8,04905	-0,0682	8,14396	-4,6847	8,24066	-77,2202	-9,80724	-157,9457	56,480	2,222190
MDesliz32	2,216655	-1.#IND00	0,545	8,05341	-0,0795	8,14480	-4,7338	8,23313	-77,7644	-9,86866	-160,0996	56,679	2,222198
MDesliz1y2_1	2,216629	-1.#IND00	4,124	8,05721	-0,0602	8,14663	-3,3160	9,61567	-64,8032	-4,04295	-164,3807	50,807	2,222288
MDesliz1y2_2	2,216615	-1.#IND00	2,921	8,05415	0,0515	8,14592	-3,0662	8,68206	-61,5405	-4,94616	-107,8263	51,636	2,222222
MDesliz1y2_4	2,216597	-1.#IND00	3,388	8,05654	-0,0139	8,14936	-3,5176	8,40849	-64,9936	-7,61975	-111,2763	55,028	2,222231
MDesliz1y2_8	2,216706	-1.#IND00	4,370	8,03819	0,0064	8,14561	-3,6510	8,28468	-67,0836	-8,84979	-116,6266	55,026	2,222167
MDesliz1y2_16	2,216657	5,74E-04	6,169	8,04064	-0,1161	8,13899	-3,7905	8,17181	-67,5282	-9,11742	-115,9941	55,065	2,222204
MDesliz1y2_32	2,216678	1,29E-03	7,003	8,05132	-0,0836	8,15925	-3,7149	8,20723	-66,7168	-8,90237	-114,6111	55,028	2,222172
MFinPer2	2,218424	-1.#IND00	2,118	8,06548	-0,0592	8,14257	-4,2449	9,22522	-73,9230	-4,99211	-196,3968	53,138	2,224009
MFinPer4	2,216901	-1.#IND00	1,149	8,05898	-0,1055	8,13996	-4,6451	9,09566	-78,1525	-6,08485	-215,0383	54,744	2,222433
MFinPer8	2,216647	-1.#IND00	0,802	8,04687	-0,1038	8,12975	-4,8433	9,04477	-80,3490	-6,29938	-225,6228	55,450	2,222379
MFinPer16	2,216794	-1.#IND00	0,653	8,05751	-0,1495	8,13204	-4,9258	9,05936	-81,4372	-6,53615	-230,7984	55,643	2,222245
MFinPer32	2,216694	-1.#IND00	0,561	8,05798	-0,1092	8,12481	-5,0200	9,04528	-82,0657	-6,51647	-233,7837	55,835	2,222245
MDesliz1y2_1_NoSoH	2,216620	-1.#IND00	0,040	8,05722	-0,1041	8,14653	-4,7442	8,56622	-78,1997	-10,95300	-161,7532	55,065	2,222271
MDesliz1y2_2_NoSoH	2,216620	-1.#IND00	2,450	8,05674	-0,0560	8,14651	-3,8717	8,41096	-69,2064	-8,20788	-125,8503	55,065	2,222213
IntegradorNoSOH	2,216624	-1.#IND00	REFEREN	8,05745	-0,1940	8,14402	-6,5612	8,01443	-96,0803	-13,40053	-233,6134	60,009	2,222201
Integrador	2,216631	-1.#IND00	0,483	8,06404	-0,1700	8,13102	-5,0325	9,05191	-82,5511	-6,53476	-236,3656	55,964	2,222220

Tabla 4.1 Resumen de resultados de las simulaciones:
indicadores de calidad de los 23 métodos de sensado

De los diagramas de Bode, se han extraído los correspondientes a 4 frecuencias representativas: 50 Hz, 1 KHz, 10 KHz y 40 KHz.

Se ha añadido un indicativo más en la columna de más a la derecha: los valores de pico de la sinusoide de cada sensador.

Para los indicativos FFT a 50 Hz, Bode a 50 Hz y Amplitud de pico, que tienen valores tan próximos, se han calculado también sus diferencias relativas respecto del patrón "IntegradorNoSOH", que se presentan en la siguiente tabla:

		Bode 50 H	Amplitud
Método de sensado	FFT (50 Hz)	mplitud (dB	de pico
LPFSimple10K	0,00009%	0,000%	-0,003%
LPFDoble20K	0,00007%	-0,001%	-0,001%
Bessel	-0,00014%	-0,001%	-0,003%
MDesliz2	-0,00011%	0,077%	0,000%
MDesliz4	0,00007%	0,016%	0,001%
MDesliz8	0,00545%	-0,203%	-0,002%
MDesliz16	0,00275%	-0,104%	0,000%
MDesliz32	0,00139%	-0,050%	0,000%
MDesliz1y2_1	0,00023%	-0,003%	0,004%
MDesliz1y2_2	-0,00041%	-0,041%	0,001%
MDesliz1y2_4	-0,00123%	-0,011%	0,001%
MDesliz1y2_8	0,00372%	-0,239%	-0,002%
MDesliz1y2_16	0,00148%	-0,209%	0,000%
MDesliz1y2_32	0,00244%	-0,076%	-0,001%
MFinPer2	0,08119%	0,100%	0,081%
MFinPer4	0,01251%	0,019%	0,010%
MFinPer8	0,00106%	-0,131%	0,008%
MFinPer16	0,00766%	0,001%	0,002%
MFinPer32	0,00315%	0,007%	0,002%
MDesliz1y2_1_NoSoH	-0,00018%	-0,003%	0,003%
MDesliz1y2_2_NoSoH	-0,00016%	-0,009%	0,001%
IntegradorNoSOH	0,00000%	0,000%	0,000%
Integrador	0,00031%	0,082%	0,001%

Tabla 4.2 Resumen de resultados de las simulaciones:
Algunos indicadores de calidad de los 23 métodos de sensado relativos.

Para facilitar la comprensión visual, a continuación se presentan los gráficos extraídos de la tabla para los diferentes indicadores, que sirven de comparativa entre los métodos de sensado evaluados:

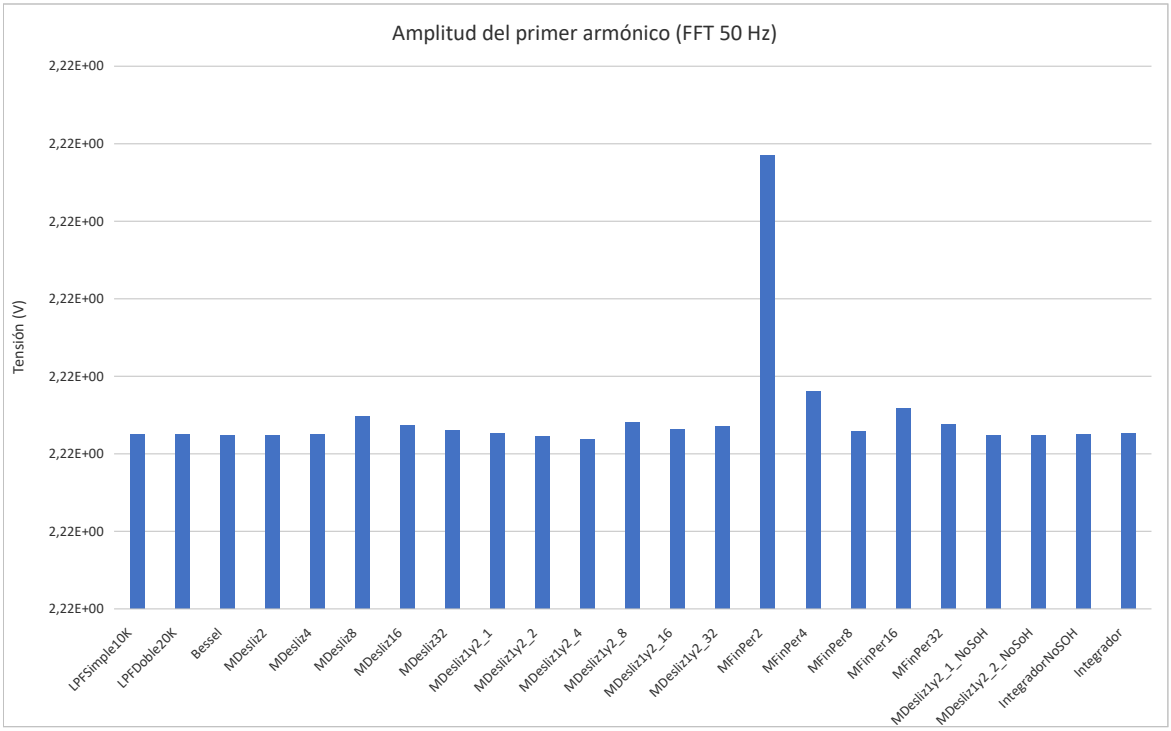


Fig. 4.8 Gráfico de amplitud del primer armónico obtenido por FFT

Como se aprecia, tanto en el gráfico como en la tabla 4.2, la diferencia relativa entre los sensados se mantiene inferior al 1 por 1000, por lo que este indicador no se considera discriminatorio para comparar los métodos.

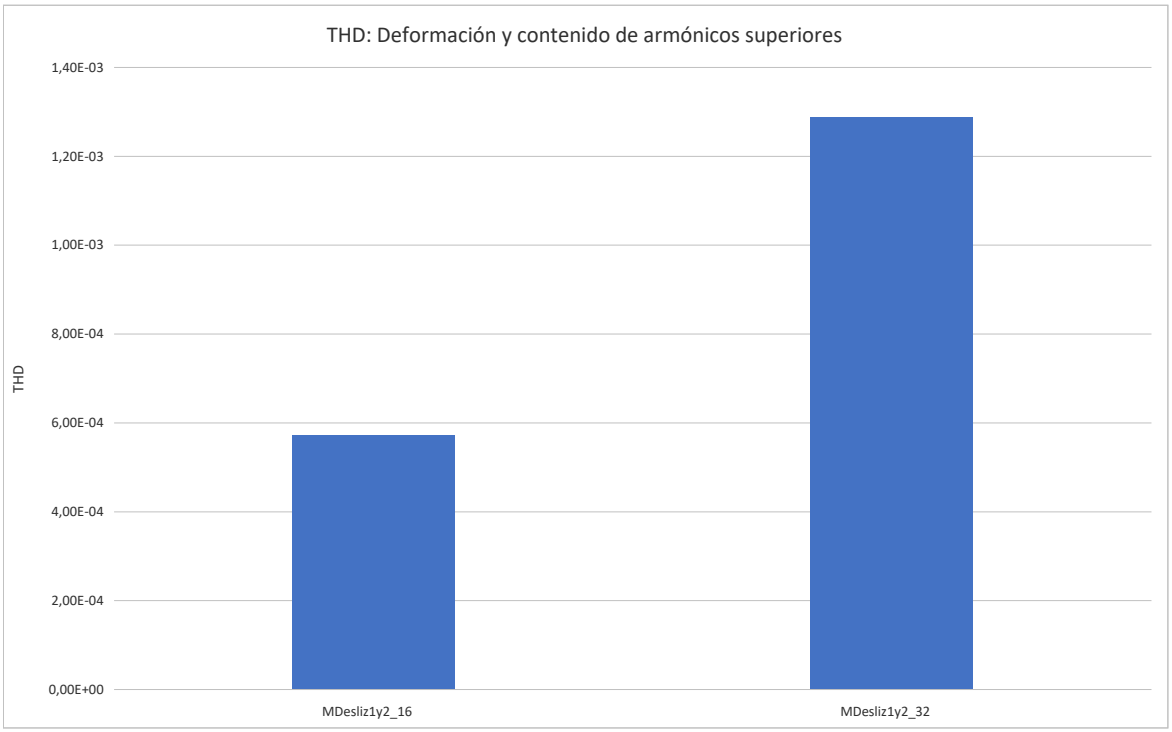


Fig. 4.9 Gráfico de la distorsión armónica total (THD)

El gráfico anterior sólo da valores para dos de los métodos de sensado, porque el resto han dado en Simview un valor de error "-1.#IND000e+000". En todo caso, de la observación detallada en Simview de las curvas en la sinusoide, se desprende que la diferencia entre ellas era mínima, por lo que cabe esperar que todas den un valor de THD similar a las presentadas e inferior al 2 por 1000, por lo que, igualmente al anterior, este indicador no se considera discriminatorio para comparar los métodos.

.

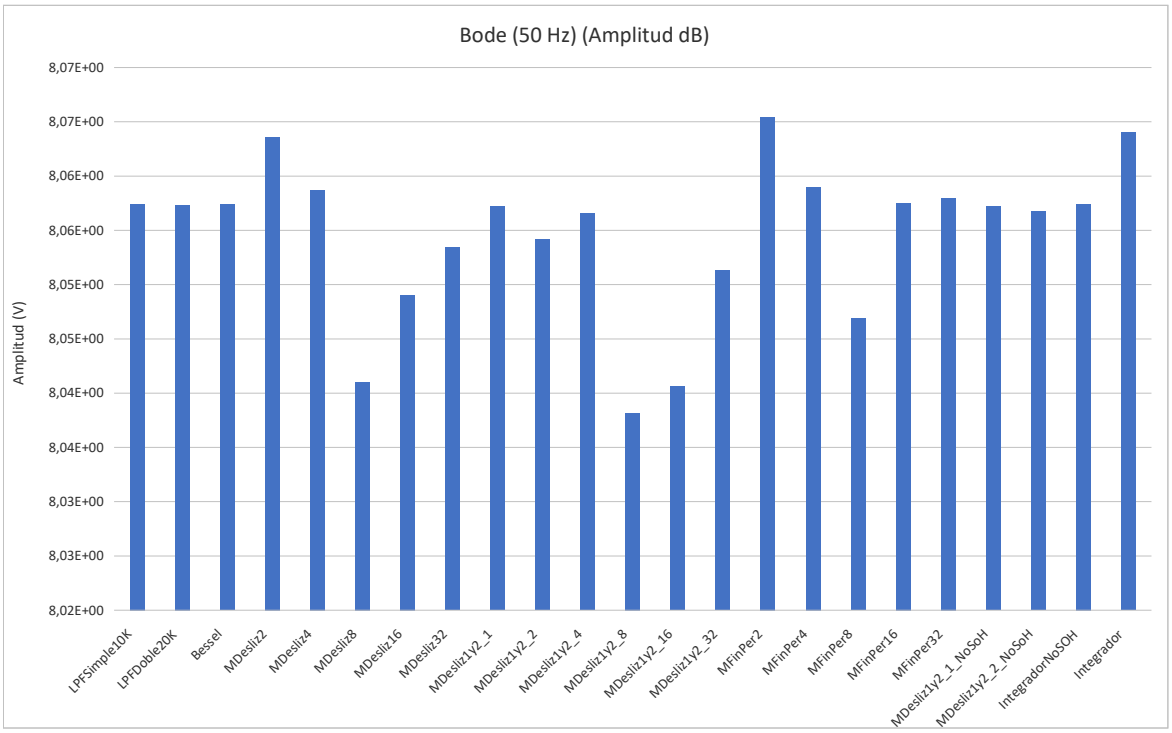


Fig. 4.10 Gráfico de amplitud del diagrama de Bode a 50 Hz

Aunque en el gráfico, por el modo como ha quedado presentado, parece advertirse cierta diferencia entre los métodos, en la tabla 4.2 se aprecia que ningún método de sensado tiene una diferencia relativa respecto al patrón mayor del patrón “IntegradorNoSOH” y que, además, el orden de esas diferencias es muy diferente del de los anteriores indicadores de módulo, por lo que este indicador tampoco se considera discriminatorio para comparar los métodos.

A continuación se presentan el resto de gráficos de amplitud del diagrama de Bode:

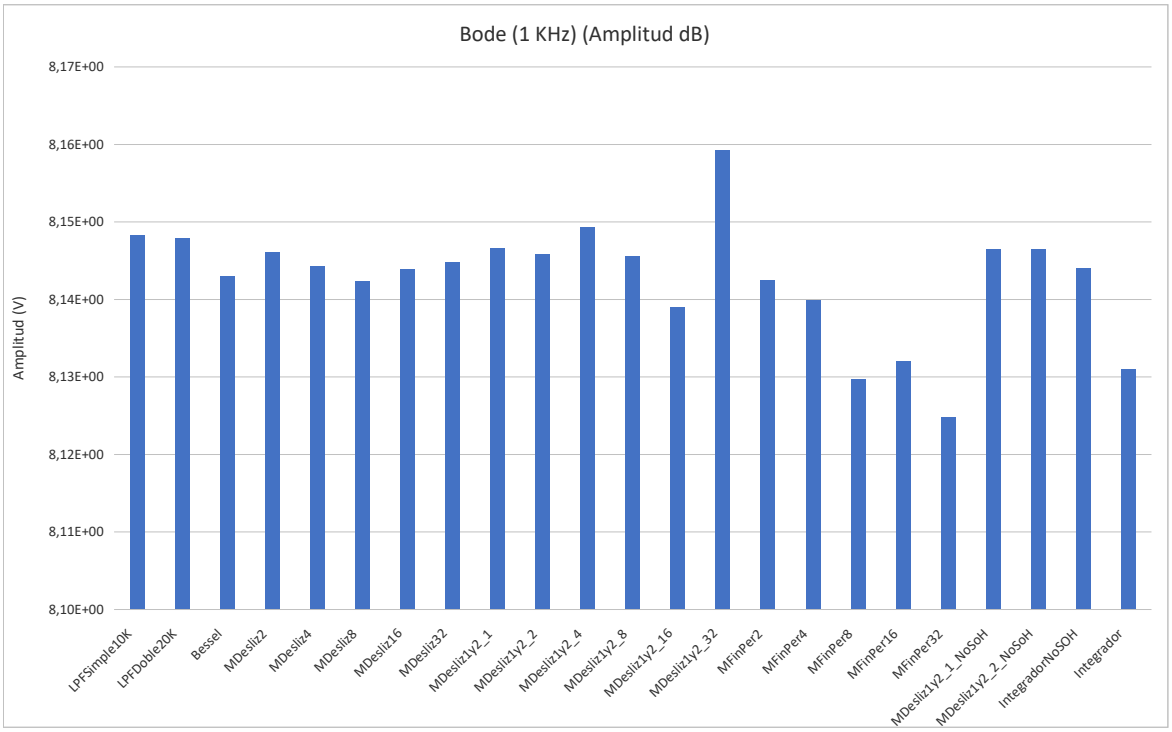


Fig. 4.11 Gráfico de amplitud del diagrama de Bode a 1 KHz

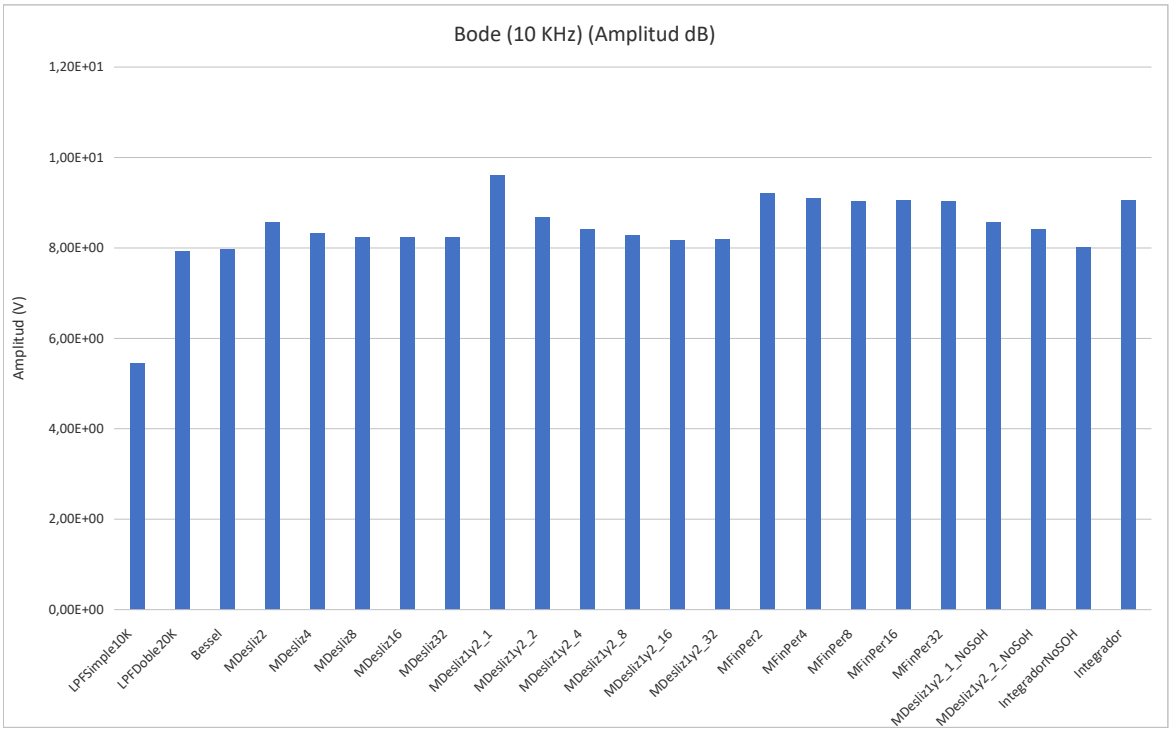


Fig. 4.12 Gráfico de amplitud del diagrama de Bode a 10 KHz

En los dos gráficos anteriores, de amplitud a 1 y 10 KHz, se observa poca caída todavía, y parecida entre ellos, salvo para el “LPFSimple10K”, que empieza a caer algo más rápido que los demás.

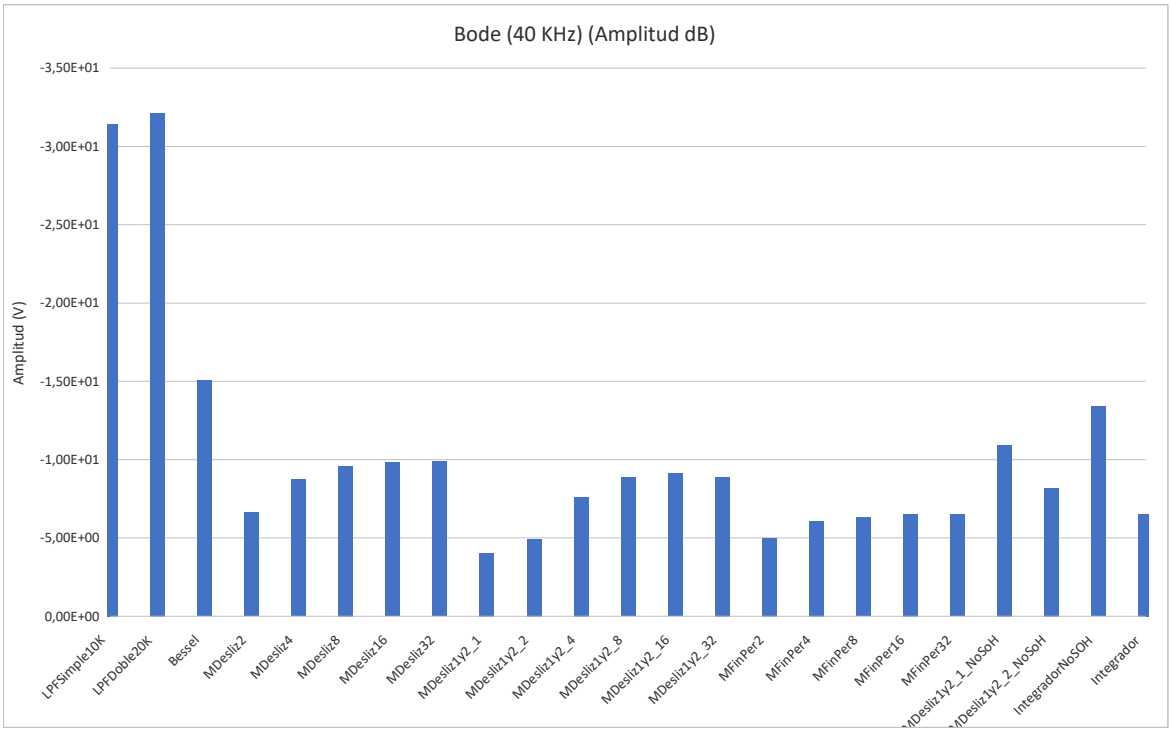


Fig. 4.13 Gráfico de amplitud del diagrama de Bode a 40 KHz (Atención: negativo)

En el gráfico anterior, de amplitud a 40 KHz, ya se observa una caída apreciable en todos, siendo los “MDesliz1y2_1” y “MDesliz1y2_2” los que tienen menor caída.

A continuación van a presentarse los gráficos de las fases de los diagramas de Bode:

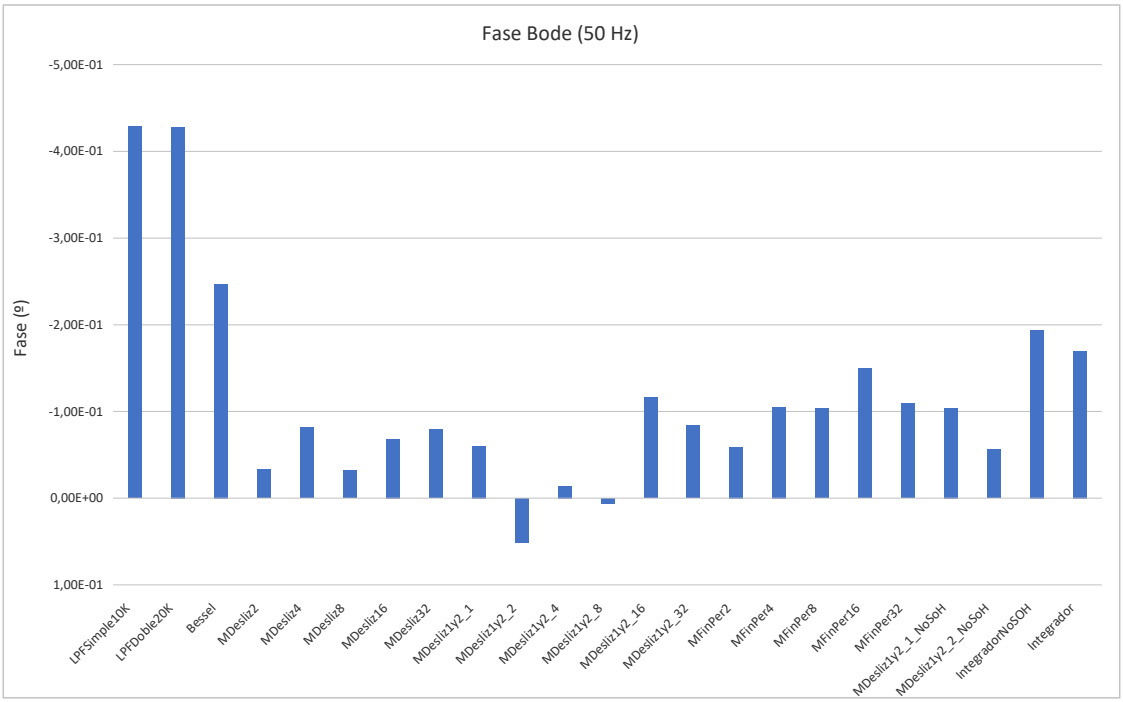


Fig. 4.14 Gráfico de fase del diagrama de Bode a 50 Hz

El anterior gráfico de fase a 50 Hz presenta valores muy pequeños que no resultan discriminatorios.

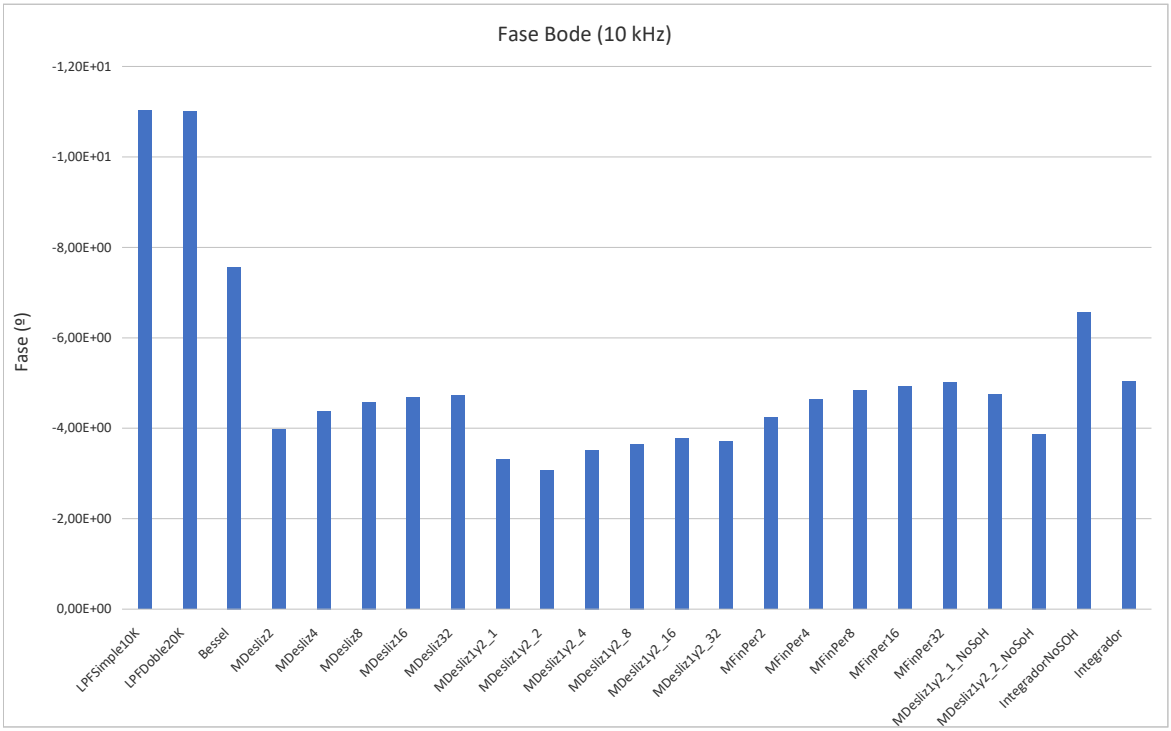


Fig. 4.15 Gráfico de fase del diagrama de Bode a 1 KHz

En el anterior gráfico de fase a 1 KHz, ya empieza a verse un desfase apreciable de los 3 filtros analógicos, así como un desfase menor respecto a los demás de los sensadores muestreadores “MDesliz1y2_1” y “MDesliz1y2_2”.

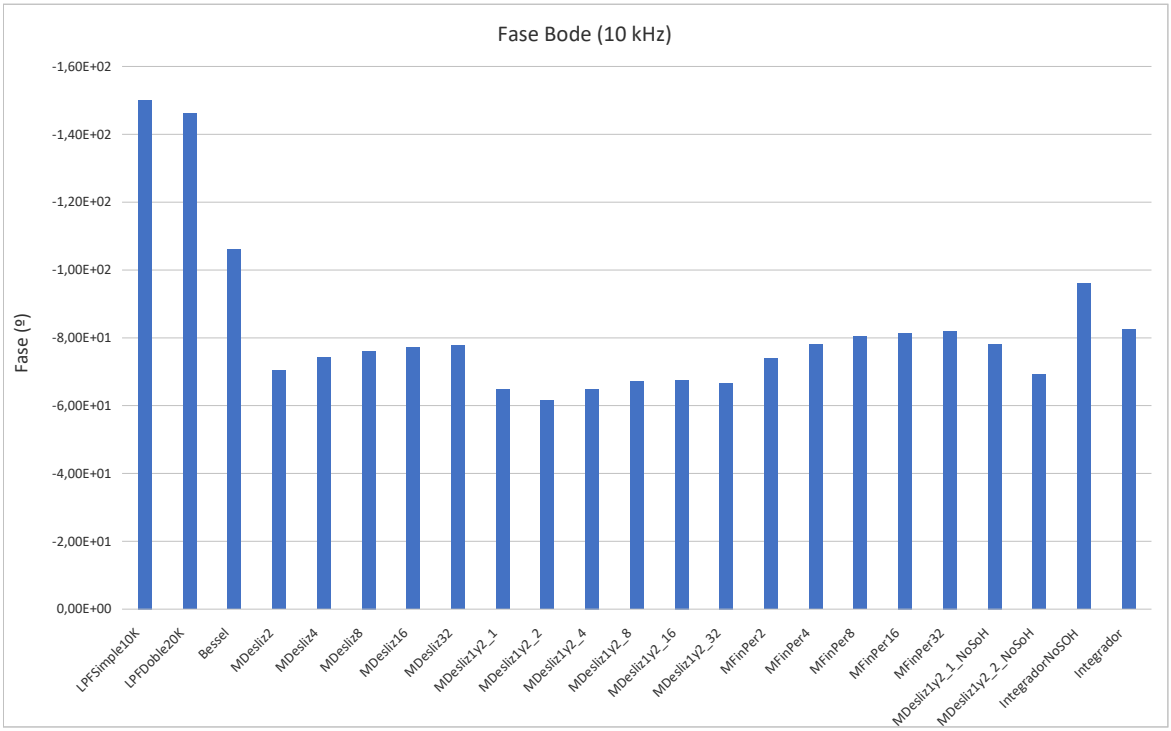


Fig. 4.16 Gráfico de fase del diagrama de Bode a 10 KHz

En el anterior gráfico de fase a 10 KHz, aparecen ya desfases importantes en todos los sensadores, pero se confirma el mayor desfase los 3 filtros analógicos, así como el desfase de los sensadores muestreadores “MDesliz1y2_1” y “MDesliz1y2_2” menor que del resto.

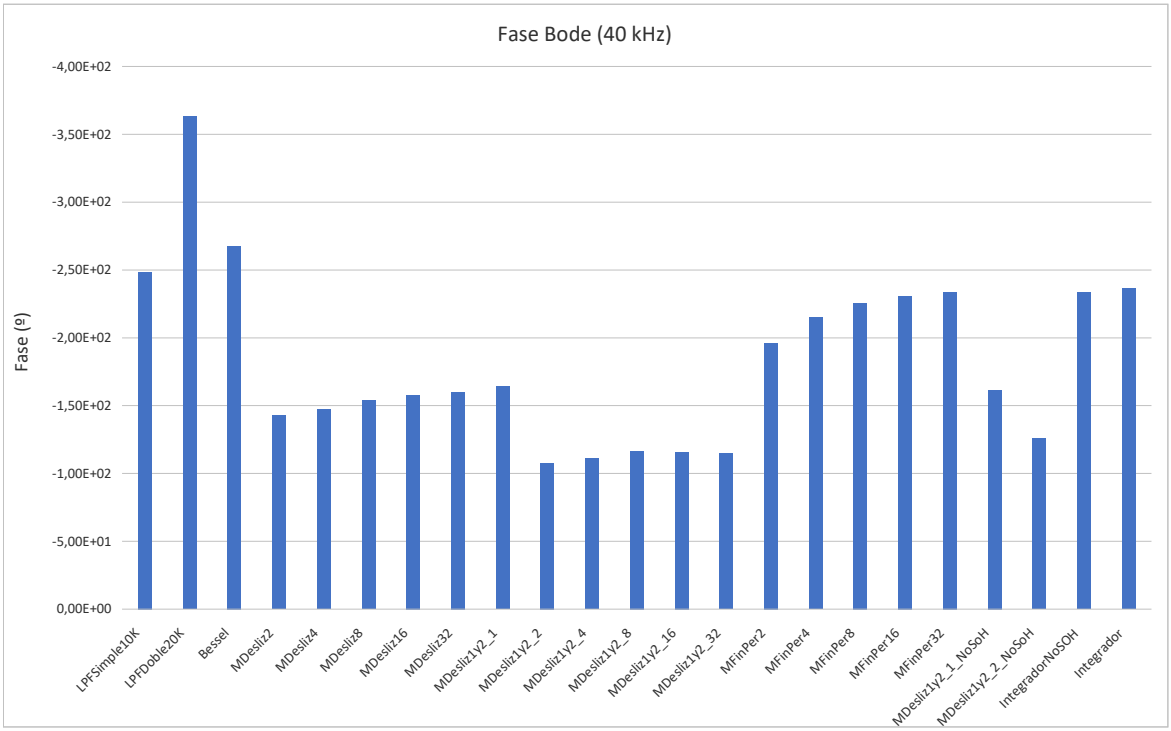


Fig. 4.17 Gráfico de fase del diagrama de Bode a 40 KHz

En el anterior gráfico de fase a 40 KHz, ya próximos a la frecuencia de modulación, los desfases son ya muy importantes en todos los sensadores y sigue confirmándose el mayor desfase los 3 filtros analógicos, así como el desfase de los sensadores muestreadores “MDesliz1y2_1” y “MDesliz1y2_2”, menor que los del resto.

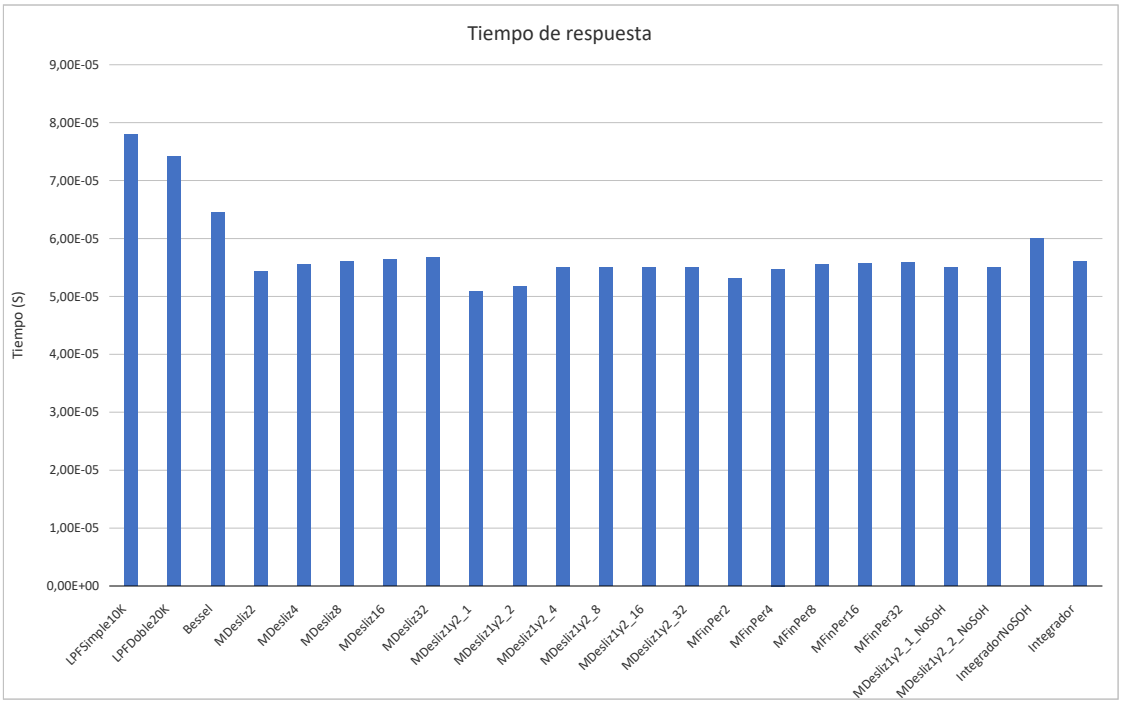


Fig. 4.18 Gráfico de tiempos de respuesta de los sensadores a un escalón de una entrada

Este gráfico es uno de los más discriminatorios y conviene examinarlo detenidamente. Hay que tener en cuenta que este tiempo de respuesta incluye el retardo, común a todos los sensadores, de 6 periodos de modulación, que tarda la corriente media en llegar al nuevo valor permanente (debido al filtro LC). Esta subida puede observarse mejor en la figura 2.12 anterior, que se repite a continuación para facilidad de consulta.

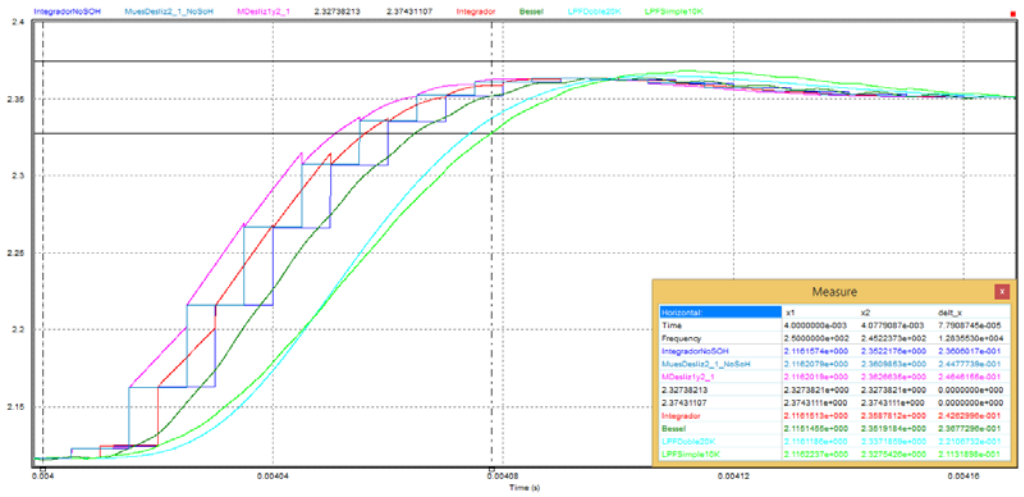


Fig. 2.12 Medida del tiempo de respuesta de varios métodos de sensado

En ella, la curva de las presentadas que mejor aproxima la corriente media en módulo y posición temporal es la de color rosa de más a la izquierda, “MDesliz1y2_1”, aunque el patrón de referencia de su módulo es la azul oscura a escalones “IntegradorNoSOH”, que está retrasada concretamente, sus vértices, los 5 µs de medio periodo de modulación.

Examinando tanto el gráfico de la figura 4.18 como la columna “Tiempo de respuesta” de la tabla Excel 4.1, puede observarse cómo, aparte de los tres sensadores de filtro analógico, que tienen un valor mayor de 64 µs, el resto puede considerase que forman 3 grandes grupos, alrededor de los 50, 55 y 60 µs.

En el grupo de 50 µs, el de los mejores sensadores, se encuentran los “MDesliz1y2_1” y “MDesliz1y2_2”, con 50.8 µs y 51.6 µs, respectivamente. Hay que advertir que esa diferencia entre ellos se ha debido a cómo ha coincidido que sus predictores SOH han efectuado sus proyecciones de evolución en ese salto en particular, pudiendo quedar en diferente orden entre ellos en otros casos.

En el grupo de 60 µs está el patrón de referencia “IntegradorNoSOH”, que da su medida al final de cada periodo de modulación.

Y en el grupo de 55 µs se encuentran el resto de los sensadores síncronos, que se comentan a continuación:

De ellos, unos, como los “MfinPer<N>”, dan su medida sólo al fin del periodo de modulación, pero el SOH interpola correctamente las transiciones entre los puntos de esas medidas y produce ese adelanto de medio periodo de modulación respecto al final de periodo.

Por otro lado, los deslizantes dan medida también en instantes intermedios del periodo de modulación, pero esas medidas intermedias van sólo progresivamente incorporando la variación en la corriente, de modo que sólo la han incorporado completa en el fin del periodo de modulación. En éstos, el SOH suaviza también las transiciones entre los puntos, de modo que quedan bastante igualados todos los deslizantes, aunque hayan dado medidas en diferente número de instantes dentro del periodo de modulación.

También están en este grupo de 55 µs los “MDesliz1y2_1_NoSOH” y “MDesliz1y2_2_NoSOH”, por dar su medida en el centro de los periodos de modulación, pero no disfrutando del suavizado del SOH.

5. PROPUESTA DE SOLUCIONES

5.1 Introducción

En el presente capítulo se van a analizar los resultados de las simulaciones y comparativas presentados en el anterior para elegir la mejor o mejores soluciones al problema estudiado, para proponer el que se considere el mejor método de sensado de corriente media en conversores de tensión con modulación PWM, válido para inversor trifásico.

5.2 Análisis de los resultados y selección de la solución

Examinando los resultados obtenidos, presentados en el capítulo anterior, se advierte que todos los métodos de sensado probados tienen una exactitud en la medida del módulo de la corriente muy satisfactoria, no existiendo ventaja significativa entre ellos, por lo que, desde el punto de vista de la exactitud de medida del módulo, todos resultan igualmente válidos.

Donde sí que se advierten diferencias apreciables entre ellos es en el tiempo de disponibilidad de la medida. Los indicadores que miden ese tiempo: las fases en el diagrama de Bode a distintas frecuencias y los tiempos de respuesta al escalón de excitación, concuerdan en dar como mejores a dos métodos de sensado: el “MDesliz1y2_1” y el “MDesliz1y2_2”, es decir, respectivamente, el que da como medida la corriente instantánea en las crestas de la señal portadora y el que la da tanto en las crestas como en los valles.

De entre ellos, el “MDesliz1y2_1” ha dado un tiempo de respuesta ligeramente mejor, pero ya se ha explicado que se ha debido a cómo ha coincidido la proyección de su SOH en la medida concreta. Sin embargo, el “MDesliz1y2_2” ha dado los desfases menores en todos los diagramas de Bode de fase, y con diferencias significativas respecto a todos los demás, incluidas sus versiones sin SOH.

Por todo lo anterior, los estudios y experimentaciones en simulación efectuadas en este trabajo **califican al "MDesliz1y2_2" como el mejor método de sensado de corriente media por una fase de todos los estudiados, por lo que es el que se propone como mejor solución al problema de sensado.**

Este método síncrono entrega como medida, en los instantes de cada cresta y cada valle de la señal portadora triangular de la modulación PWM, la corriente instantánea por una fase muestreada en esos instantes y, entre ellos, una proyección a partir de muestras anteriores, generada por un algoritmo retenedor de segundo orden ("Second Order Hold", "SOH") modificado en cuanto a restringir su salida, limitándola con la de un retenedor de primer orden e impidiendo su retroceso.

En segunda clara posición ha resultado el "MDesliz1y2_1", que entrega la muestra sólo en las crestas y también suavizando las transiciones con un SOH.

Entre estos dos, además de la ventaja clara en los indicadores de calidad establecidos, merece la pena observar que el sensado elegido tiene la ventaja de que se ajusta a la corriente muestreada en 2 instantes en cada periodo de control, con lo que su trayectoria sigue más fielmente la de la corriente media. Sin embargo, tiene el inconveniente de que requiere del elemento que se utilice para muestrear una velocidad doble que el "MDesliz1y2_1". Caso de ser esto una limitación para el hardware, se optaría por este último método de sensado, clasificado en segundo lugar.

Debe advertirse también que dichos dos métodos de sensado han funcionado tan bien en condiciones de frecuencia de cruce del filtro LC menor que la décima parte de la frecuencia de modulación, lo que hace que las trayectorias de la corriente instantánea sean muy rectas. En otras condiciones, cabría la posibilidad de que las comparativas dieran clasificaciones diferentes. Sin embargo, se considera que dicha condición debería darse siempre en inversores bien diseñados, porque es necesaria para un correcto filtrado de la corriente de salida que permita al inversor entregar una tensión suficientemente limpia de rizado de la modulación.

Por otro lado, como se ha explicado anteriormente, las simulaciones de este estudio se han realizado sin imponer tiempos muertos a los conmutadores. En una aplicación real, los tiempos muertos en el modulador PWM son necesarios y, si se implementan produciendo

un adelanto de los ceses de conducción de los conmutadores y un retraso igual de los pases a conducción, podrían suponer cierta asimetría en las rampas de corriente por las fases que perjudicara la cancelación o compensación entre las muestras de los métodos de sensado por muestreo y cierto desplazamiento temporal del instante en que la corriente instantánea iguala a la corriente media en todo el periodo. Aunque no es objeto del presente trabajo, se apunta aquí que un posible modo de solucionarlo sería dejar en su instante nominal (sin adelantar) las aperturas (ceses de conducción) de los conmutadores y sólo retrasar sus cierres (pases a conducción), dejando que los diodos en antiparalelo conduzcan las corrientes presentes en las inductancias del filtro de salida en el instante adecuado (tras la apertura del conmutador opuesto) hasta que el correspondiente conmutador pase a conducción, tras el tiempo muerto. No se entra en más detalle por no ser los tiempos muertos objeto de este estudio.

5.3 Ventajas de la solución

La solución de sensado elegida presenta la ventaja principal de su **mínimo tiempo de respuesta**. Adelanta aproximadamente en medio periodo de conmutación respecto al resto. La implicación práctica es que va a permitir diseñar el control del convertidor PWM con mayor rapidez de respuesta, con lo que va a poder oponerse con mayor rapidez y eficacia a las perturbaciones que puedan surgir: en la tensión de alimentación Vdc y en la carga. Ello supondrá una ventaja de calidad del convertidor que lo incorpore respecto a su competencia.

6. VALIDACIÓN DE LAS SOLUCIONES

6.1 Introducción

Para validar la solución propuesta, se realiza una comprobación de funcionamiento del inversor trifásico incorporándola, con el lazo de control de corriente cerrado y con unas perturbaciones de Vdc y de carga, para comprobar el comportamiento del control con ellas.

Para ello, primeramente se obtiene un modelo de pequeña señal del inversor a controlar provisto del sensador a verificar.

Para diseñar el controlador se aprovecha una herramienta software muy potente, complementaria al PSIM, llamada “SmartCtrl”.

En principio, cada sensador puede tener diferente respuesta, por lo que se requeriría un controlador diferente para cada uno.

Se realizan simulaciones en PSIM con el controlador diseñado y se evalúa su comportamiento.

6.2 Modelo de pequeña señal

Normalmente, para diseñar el controlador, se desarrollaría un modelo de pequeña señal del sistema a controlar, se generaría un diagrama de Bode del mismo, que se utilizaría para aplicarle los métodos clásicos de diseño de controladores.

Sin embargo, gracias a la herramienta “SmartControl”, complementaria del PSIM, ello no ha sido necesario, ya que nos proporciona directamente tanto dicho diagrama de Bode como un método interactivo cómodo para diseñar el control, que se ha utilizado como indica el punto siguiente.

6.3 Diseño del control de corriente

Se va a detallar en este punto el diseño del controlador para la solución de sensado elegida.

Aunque el inversor está definido como una fuente de tensión trifásica, que es el tipo más demandado, como se ha explicado

Se va a diseñar un control de la planta mediante el uso de la herramienta SmartCtrl indicada. Esta herramienta necesita importar un diagrama de Bode de la señal a controlar (en este caso, Vef), el cual se obtiene haciendo un barrido de frecuencias con el PSIM, gracias a una potente utilidad que ofrece, denominada “AC Sweep”.

Antes del barrido, es necesario adaptar ciertos parámetros de la simulación que pueden afectarle.

Para que las sinusoides moduladoras no mezclen su efecto, se cambia su frecuencia a 0.001 Hz.

Para que el punto de operación no tenga ninguna tensión de fase a 0, se cambia el valor de la fase a 90º.

Además, se desactivan las perturbaciones.

Antes de iniciar el barrido, conviene también comprobar la respuesta de la planta con los elementos y los valores de parámetros que se vayan a utilizar en dicho barrido. Para ello, se ajusta el valor de pico de la sinusoidal de entrada a 0.2 V, se realiza la simulación y se obtiene la siguiente respuesta, que se ve correcta:

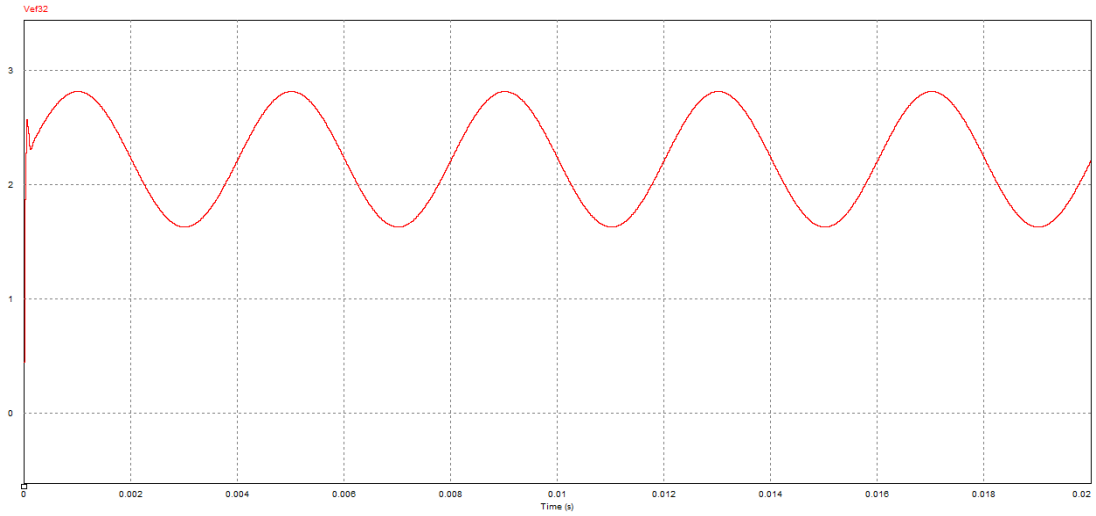


Fig. 6.1 Respuesta del inversor a entrada del barrido AC Sweep

Una vez comprobada la respuesta, se va a proceder con el barrido, para lo que se ajustan en el elemento ACSWEEP los parámetros del barrido:

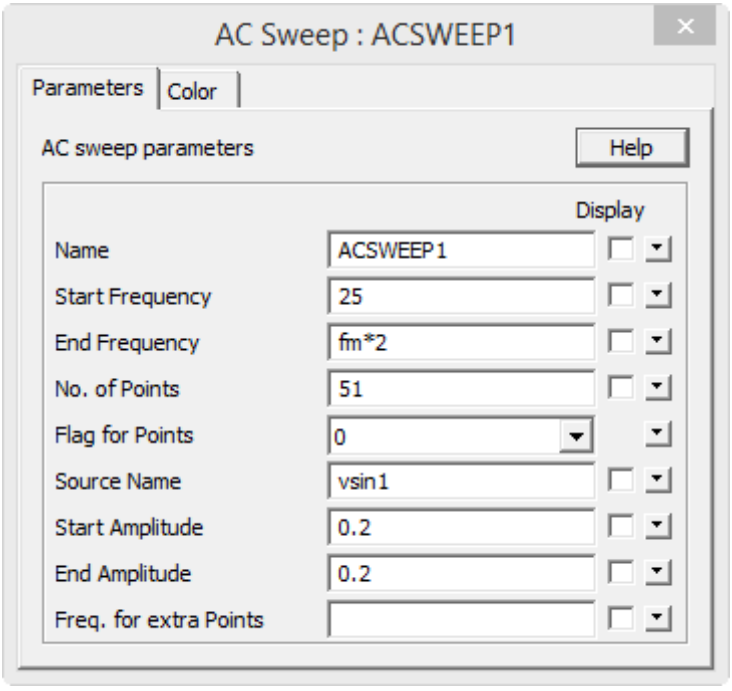


Fig. 6.2 Ajuste de parámetros del barrido AC Sweep

-Se efectuará el barrido de frecuencia de 25 Hz a 200 kHz para cubrir todo el rango de interés.

-51 puntos para lograr una precisión decente.

-El “Flag for Points” se pone a cero para presentar en escala logarítmica.

-El nombre de la fuente se hace coincidir con el del generador de la sinusoide de entrada.

-Se podría hacer una progresión de la amplitud de entrada, pero en este caso, para no llegar a los límites del limitador de amplitud del **am**, se ajusta una “End Amplitude” igual a la “Start Amplitude”

La simulación con este barrido tarda bastante tiempo. Existe una opción más rápida utilizando el elemento “AC Sweep (multisine)”, pero pierde algo de precisión y no se va a utilizar ahora.

Ejecutada la simulación, se obtiene el siguiente diagrama de Bode:

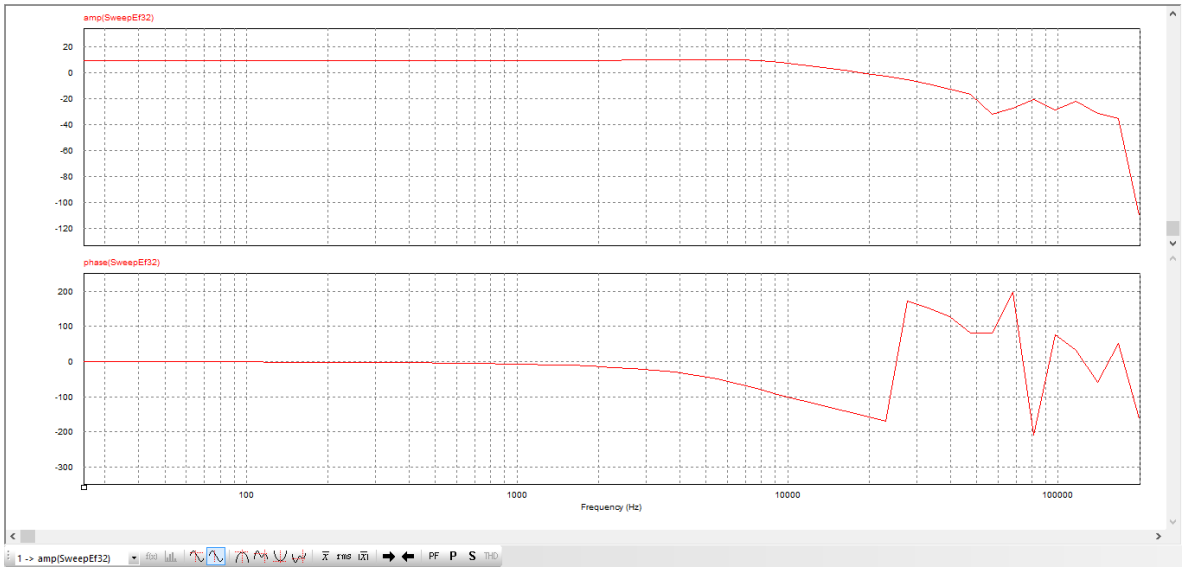


Fig. 6.3 Diagrama de Bode del inversor en lazo abierto.

El inesperado ascenso de la fase hacia la frecuencia de 25 KHz se debe a un error de este tipo de barrido del PSIM: Cuando la respuesta de la fase cae por debajo de los -180º a veces suma 360º a los valores siguientes. Cuando esto ocurre, hay que corregir estos errores antes de importar los datos en el SmartCtrl para que pueda funcionar bien. Un modo de hacerlo es guardar los resultados como fichero txt, abrirlo en Excel y corregirlos, restándoles 360º, y volver a guardarlos como .txt para su importación en el SmartCtrl.

Para importarlos en el SmartCtrl, se selecciona la opción “File / New and initial dialog”. En la pestaña que aparece, que se presenta en la figura siguiente, se pulsa la opción “Import frequency response data from txt file” y se carga el fichero de texto corregido.

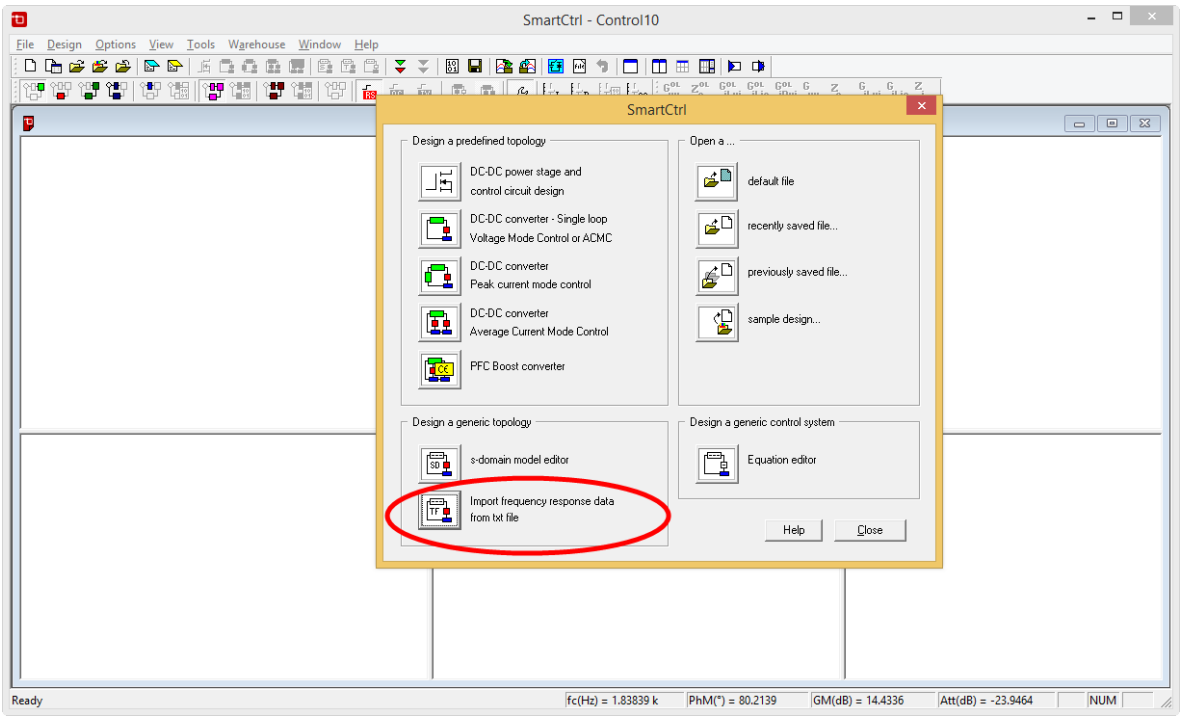


Fig. 6.4 Selección de opción de Importación del fichero de texto al SmartCtrl.

En la ventana que presenta a continuación, se introduce un valor Vo igual al valor medio de la señal Vef a controlar, que se ha observado anteriormente en la figura 6.1.

En la pantalla que SmartCtrl presenta a continuación, se selecciona el sensor “Isolated V. sensor”,

La pantalla que SmartCtrl presenta a continuación, que se muestra en la figura siguiente, permite ajustar parámetros del sensor. Se ajusta un valor de Vref muy similar al Vo, lo que produce ganancia unidad, y un polo en 5 MHz para evitar las extrañas oscilaciones que da cuando se deja a 500 G:

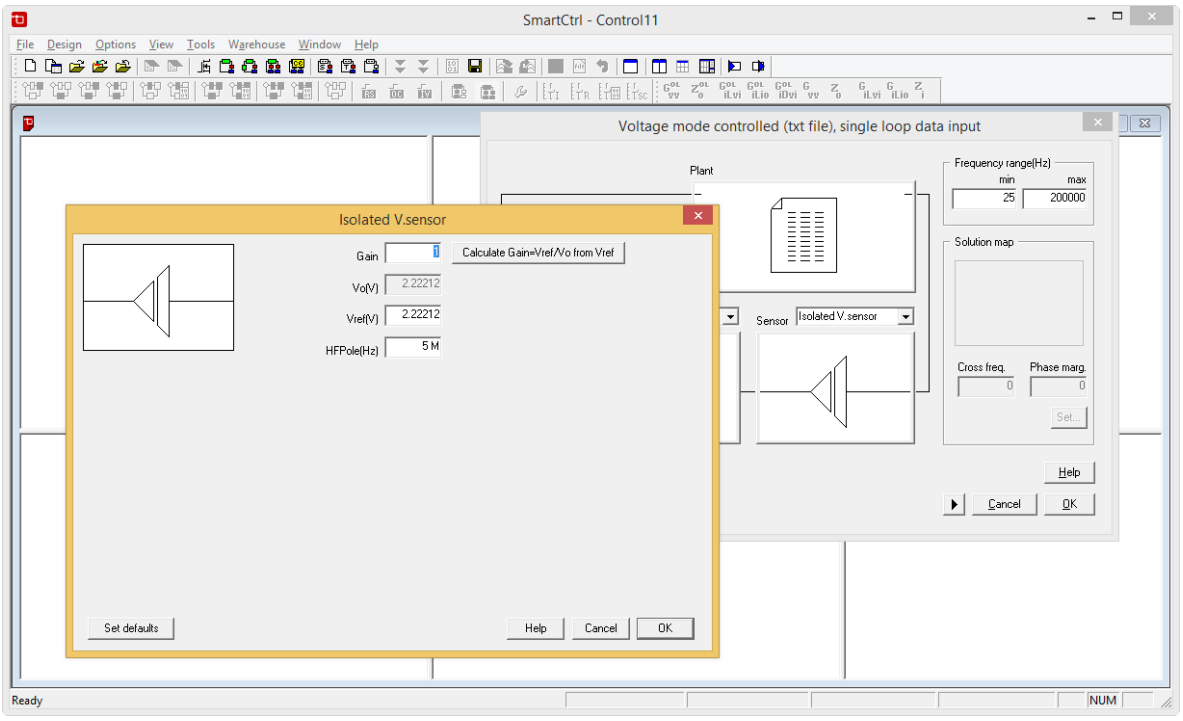


Fig. 6.5 Parámetros del sensor del controlador.

Posteriormente, para el controlador, se selecciona un tipo 3, que es el apropiado para plantas de orden 2, como es ésta (por la inductancia y el condensador del filtro). Los valores solicitados en la ventana que aparece no son relevantes porque sirven para la generación de una portadora triangular, que no se va a utilizar.

En el panel “Solution map”, se pulsa en “Set” y se selecciona un punto cualquiera del gráfico dentro de los valores admitidos (porque se podrán afinar después) y se pulsa “Ok” en las dos ventanas, como se indica en la siguiente figura:

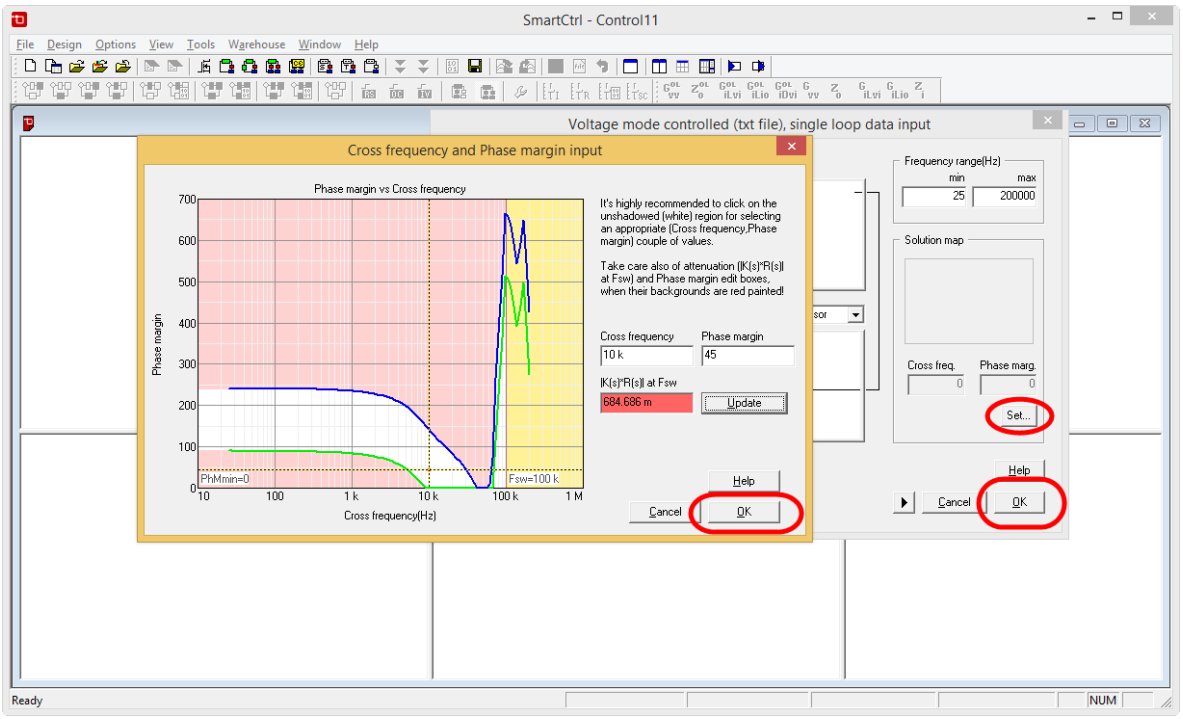


Fig. 6.6 Tanteo inicial de punto de trabajo.

En la pestaña que aparece a continuación, en “Solution map control”, que se presenta en la figura siguiente, se selecciona un punto con unas frecuencia de corte y margen de fase que produzcan una buena respuesta al escalón (representada en el cuadro de abajo).

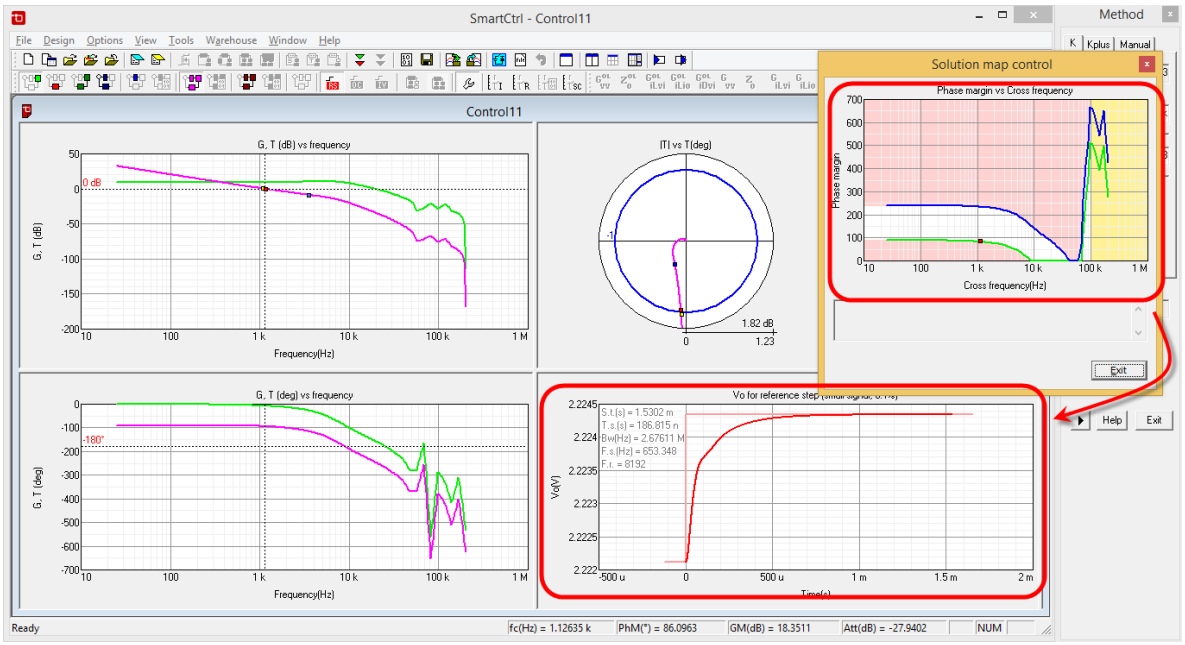


Fig. 6.7 Selección de punto de trabajo en el “Solution Map”.

Se ha elegido una respuesta algo rápida y sin sobreoscilación, que habrá que comprobar después en una simulación en PSIM.

Una vez seguidos estos pasos, ya se tienen los elementos del controlador calculados y se procede a exportarlos al PSIM. Para ello, se selecciona File/Export/To PSIM/Schematic, y se guardan seleccionando el fichero de esquema .psimsch con el que se está realizando la simulación en el PSIM.

A continuación, aparecerá en el esquema del PSIM, como se muestra en la siguiente figura, un operacional con resistencias y condensadores que implementan el controlador, más una circuitería preparada para generar la modulación, la rodeada por el rectángulo rojo, que se borra porque no se va a utilizar aquí:

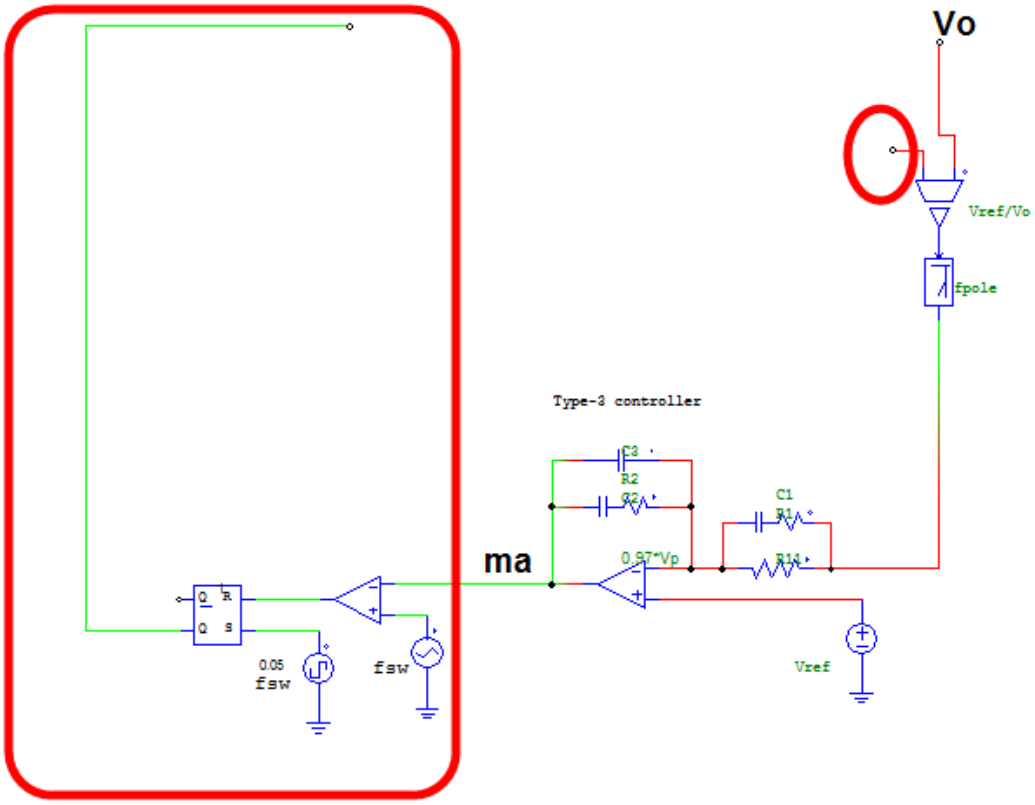


Fig. 6.8 Circuito con controlador que SmartCtrl transfiere al esquema del PSIM

En el terminal rodeado por un círculo rojo, se coloca un elemento GND y a la entrada Vo se le conecta la salida de medida a controlar, Vef.

La salida Vef se genera mediante operadores aritméticos del PSIM que calculan la raíz cuadrada de la suma de cuadrados de las corrientes de cada fase. De ese modo, se obtiene una señal proporcional a la corriente eficaz de salida que permite controlar la corriente más fácilmente.

La salida del operacional se conecta a la entrada de control de la modulación, “ma”, en lugar de la fuente “vsin1” que se utilizó para el barrido.

Con ello, queda concluido el diseño del control y aplicado al sistema a controlar. En la figura siguiente se presenta el esquema completo del inversor, con el método de sensado implementado en el bloque C y el SOH:

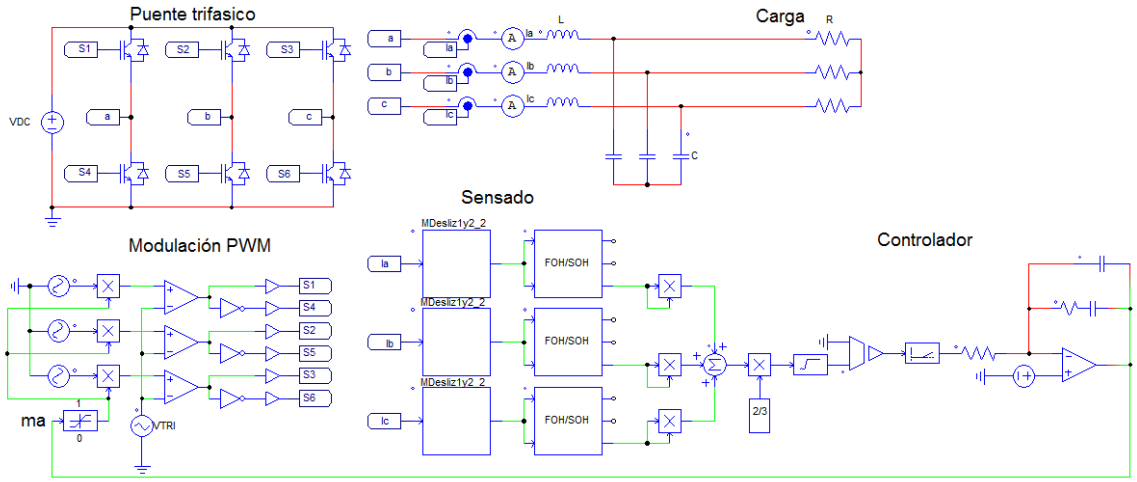


Fig. 6.9 Esquema del sistema controlado utilizando el método de sensado elegido

6.4 Evaluación del control: respuesta a perturbaciones

Efectuando la simulación, que aplica escalones de perturbaciones de Vdc y Rcarga, se obtiene la siguiente respuesta de la tensión de salida y de corriente por la inductancia:

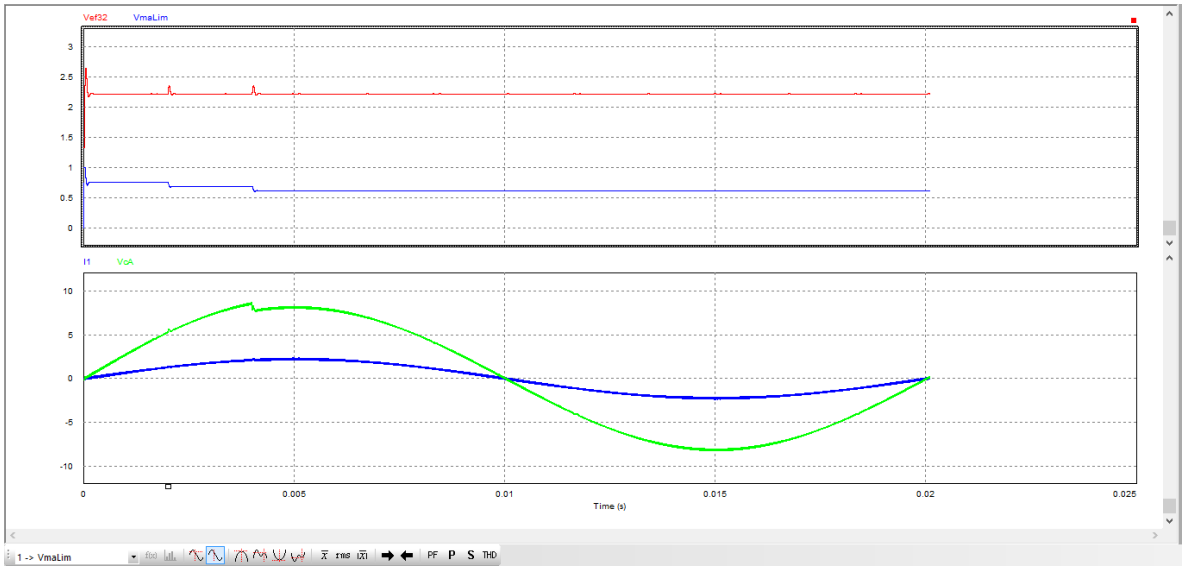


Fig. 6.10 Respuesta a perturbaciones de Vdc y Rcarga

Puede observarse cómo la tensión (la sinusoide verde) se reduce ante una reducción brusca de la resistencia de la carga, pero la corriente (sinusoide azul) sigue regulada y limpia, sin acusar la perturbación.

Se aprecia que la respuesta es rápida y sin sobreoscilaciones, gracias a la utilización del método de sensado seleccionado.

Con ello, se considera validado dicho método de sensado en este inversor trifásico y el control diseñado.

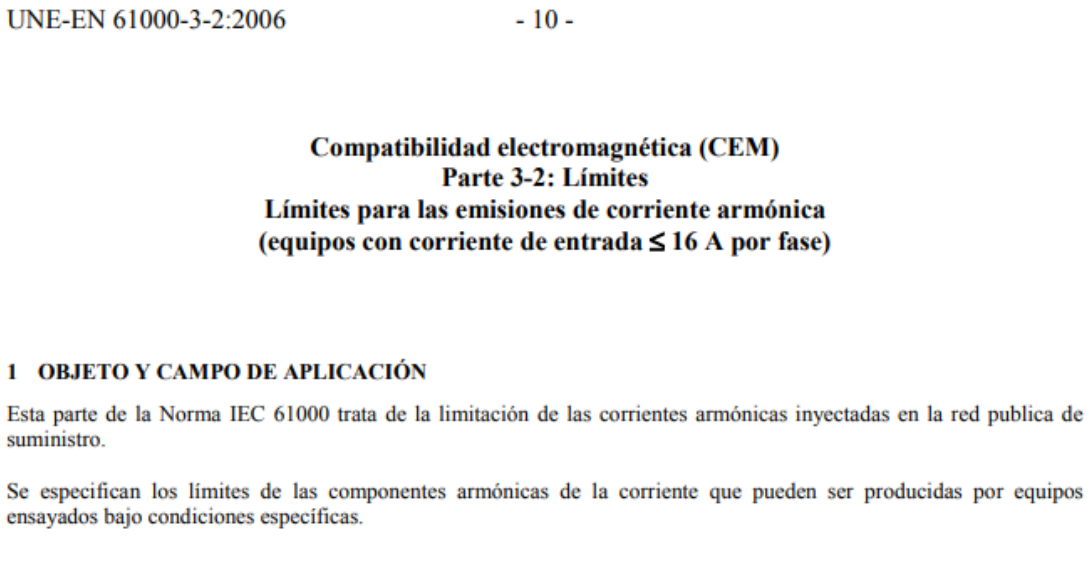
7. MARCO REGULADOR

Las labores de este trabajo se han desarrollado en simulación y están orientadas a la investigación científica, por lo que no están sujetas a regulación legal.

Sin embargo, las aplicaciones a las que podría destinarse sí deberían cumplir las normativas correspondientes a su cambo de aplicación.

Las aplicaciones principales son la conversión CC-CA para redes eléctricas aisladas en trenes, aviones o barcos; y el control de motores.

Estas aplicaciones deben cumplir la normativa que regula los armónicos de corriente, la EN 61000-3-2:



UNE-EN 61000-3-2:2006 - 10 -

Compatibilidad electromagnética (CEM)
Parte 3-2: Límites
Límites para las emisiones de corriente armónica
(equipos con corriente de entrada ≤ 16 A por fase)

1 OBJETO Y CAMPO DE APLICACIÓN

Esta parte de la Norma IEC 61000 trata de la limitación de las corrientes armónicas inyectadas en la red publica de suministro.

Se especifican los límites de las componentes armónicas de la corriente que pueden ser producidas por equipos ensayados bajo condiciones específicas.

Fig. 7.1 Extracto normativa 61000-3-2 [9]

Todos los diseños de dispositivos eléctricos o electrónicos deben cumplir las normativas de Compatibilidad Electromagnética (Directiva 2004/108/CE) y Seguridad Eléctrica:

CAPÍTULO I

DISPOSICIONES GENERALES

Artículo 1

Objeto y ámbito de aplicación

1. La presente Directiva regula la compatibilidad electromagnética de los equipos. Busca garantizar el funcionamiento del mercado interior exigiendo que los equipos cumplan un nivel adecuado de compatibilidad electromagnética. La presente Directiva se aplica a los equipos, tal como se definen en el artículo 2.

2. La presente Directiva no se aplicará a:

a) los equipos cubiertos por la Directiva 1999/5/CE;

b) los productos, componentes y equipos aeronáuticos mencionados en el Reglamento (CE) no 1592/2002 del Parlamento Europeo y del Consejo, de 15 de julio de 2002, sobre normas comunes en el ámbito de la aviación civil y por el que se crea una Agencia Europea de Seguridad Aérea (1);

c) los equipos de radio utilizados por radioaficionados, en el sentido del Reglamento de Radiocomunicaciones adoptado en el marco de la Constitución y el Convenio de la UIT (2), salvo que los equipos sean comercializados. No se considerarán equipos comercializados los kits de componentes para ser montados por radioaficionados y los equipos comerciales modificados por y para el uso de estos radioaficionados.

3. La presente Directiva no se aplicará a los equipos cuyas características físicas sean tales que:

a) no puedan generar o contribuir a las emisiones electromagnéticas que superen un nivel que permita a los equipos de radio y de telecomunicaciones, y a otros equipos, funcionar de la forma prevista; y

b) funcionen sin una degradación inaceptable en presencia de perturbaciones electromagnéticas normales derivadas de su uso previsto.

4. Cuando, en el caso de uno de los equipos a que se refiere el apartado 1, haya otras directivas comunitarias que regulen de una forma más específica todos o parte de los requisitos esenciales considerados en el Anexo I, la presente Directiva no se aplicará, o dejará de aplicarse, a ese equipo en lo que respecta a dichos requisitos a partir de la fecha de aplicación de las citadas directivas.

5. La presente Directiva no afectará a la aplicación de la legislación comunitaria o nacional que rige la seguridad de los equipos.

Fig. 7.2 Extracto Directiva 2004/108/CE del Parlamento Europeo y del Consejo [10]

8. ENTORNO SOCIO-ECONÓMICO

8.1. Presupuesto

Todos los gastos relacionados con el trabajo se detallan en la tabla de presupuestos.

En dicha tabla se detallan los gastos derivados de las horas de trabajo de los implicados en

CÓDIGO	UNIDAD	DESCRIPCIÓN	MEDICIÓN	P.U.	P.T.
		CAPÍTULO I: PERSONAL DE TRABAJO			
1.01	hora	**Tiempo de trabajo del alumno** Tiempo dedicado al desarrollo del TFG.	300	10€	3000€
1.02	hora	**Tiempo de trabajo del tutor universitario** Tiempo asociado a la orientación y preparación del TFG.	20	40€	800€
1.03	hora	**Tiempo de trabajo del tutor de empresa** Tiempo utilizado en el asesoramiento y puesta en marcha del TFG.	50	40€	2000€
				TOTAL	5800€

él:

Tabla 8.1 Gastos derivados de las horas de trabajo

Al tratarse de un trabajo de investigación teórico y soportado sobre simulación software, no ha requerido inversión en material experimental. Sin embargo, se ha precisado de software específico para su desarrollo:

CÓDIGO	UNIDAD	DESCRIPCIÓN	MEDICIÓN	P.U.	P.T.
		CAPÍTULO II: SOFTWARE			
2.01	ud	**Licencia PSIM** Software de simulación de circuitos electrónicos.	1	0€	0€
2.02	ud	**Licencia SmartCtrl** Software para diseño de control.	1	0€	0€
2.03	ud	**Licencia MATLAB** Herramienta de software para cálculos matemáticos.	1	0€	0€
				TOTAL	0€

Tabla 8.2 Gastos de software

Como se indica, los gastos derivados del software han sido nulos, gracias al uso de licencias de la universidad o de prueba, abaratando considerablemente la inversión en este trabajo.

Igualmente, la consulta de libros y artículos ha resultado gratuita, gracias a los préstamos por parte de los tutores y a la universidad, por su acceso gratuito al IEEE. No se adjunta una tabla de los escritos utilizados por estar relacionados más adelante, en la bibliografía.

Finalmente, el coste total de este TFG resulta de Cinco Mil Ochocientos euros. Es un coste relativamente bajo, gracias al acceso gratuito al material necesario y a haber podido realizarse sobre simulaciones software.

8.2. Impacto socio-económico

La aplicación de un buen sensado mejoraría la calidad de los inversores trifásicos comerciales, especialmente en cuanto a su inmunidad frente a perturbaciones. Podría suponer un incremento en la competitividad y, por tanto, en las ventas de la empresa que lo aprovechara, una mejora en la satisfacción de los usuarios finales y una reducción de las averías de los equipos alimentados mediante los inversores, por su mejor control de la tensión entregada a la salida.

Un buen control de la conversión de energía es fundamental para mejor su eficiencia, lo que aportaría un ahorro energético y contribuiría a la preservación del medio ambiente.

Aunque todos estos aspectos resultan de muy difícil cuantificación, cabe esperar de ellos un impacto muy positivo en el entorno económico.

Además, podría servir de base para futuros trabajos de investigación de otros aspectos relacionados, como el control o la fiabilidad de equipos conversores de energía, así como de otro tipo de equipos que utilicen modulación PWM.

9. CONCLUSIONES Y TRABAJOS FUTUROS

9.1. Conclusiones

Un sensado de calidad que proporcione al controlador de un convertidor de potencia conmutado por modulación de pulsos PWM una medida exacta, con el mínimo retardo y limpia, es imprescindible para un control de calidad, el cual, a su vez, incide muy notablemente en la calidad total del convertidor. Sin embargo, a pesar de los continuos avances de la tecnología de potencia y de procesado, los métodos de sensado utilizados habitualmente presentan inconvenientes difíciles de salvar sin tener que recurrir a elementos hardware de un coste excesivo.

Especial dificultad presenta el control de un inversor de tensión alterna trifásica, por la complejidad derivada de la gran interacción entre las tres fases.

En el presente trabajo, se ha estudiado y analizado distintas estrategias de sensado, evaluado y comparado su comportamiento y se ha seleccionado y propuesto aquella que permite obtener una correcta medición con una mínima caída de fase y que requiere un mínimo procesado o filtrado, que es realizable, con un coste razonable, con los recursos hardware y software disponibles en la actualidad. Por tanto, se considera conseguido el objetivo 1, general y principal, así como los subobjetivos 2 y 4.

Se han implementado mediante programas en lenguaje C las estrategias aplicables, y se ha medido y comparado en simulación sus características, de cara al objetivo de permitir un control de buena calidad, con lo que se considera cubiertos los subobjetivo 3 y 6.

Se ha validado en simulación el método de sensado propuesto, aplicándolo a las fases de un inversor trifásico por modulación de ancho de pulso PWM de portadora triangular, se ha desarrollado un controlador de corriente que lo utiliza y se ha verificado su comportamiento efectivo frente a perturbaciones. Con ello, el subobjetivo 5 se considera alcanzado.

El presente trabajo ha resultado muy formativo, porque ha requerido el repaso de conceptos estudiados en múltiples asignaturas de la carrera, como las de electrónica de potencia, control, simulación de sistemas dinámicos, instrumentación, electrónica digital, programación, su aplicación a un caso práctico muy interesante y de actualidad, así como

un proceso creativo de ingeniería consistente en estudiar en profundidad un problema y desarrollar una solución innovadora para él.

9.2. Trabajos futuros

Podría estudiarse la aplicación del método de sensado elegido a otros tipos de convertidores basados en modulación PWM, así como a cargas inductivas y con componente de tensión alterna.

Un punto fundamental para el buen funcionamiento de la solución de sensado propuesta es una correcta sincronización del sensado con el periodo de modulación. Para corregir posibles errores o desviaciones de esa sincronización, podrían investigarse mecanismos adaptativos, por ejemplo, mediante un lazo de control más lento, que ajustaran en tiempo real los instantes de muestreo para obtener la mayor coincidencia con la corriente en el periodo de muestreo.

La solución propuesta podría aplicarse y comprobarse en equipos reales, tanto a inversores de pequeña como de gran potencia.

Complementario a este estudio, podría abordarse el desarrollo de algoritmos de control que aprovechen la calidad de sensado conseguida.

10. BIBLIOGRAFÍA Y REFERENCIAS

[1] A. Barrado, Sistemas electrónicos de Potencia, trasparencias Universidad Carlos III de Madrid, 2016.

[2] F. Briz, D. Díaz-Reigosa, M. W. Degner, P. García, y J. M. Guerrero, "Current Sampling and Measurement in PWM Operated AC Drives and Power Converters", presentada en The 2010 International Power Electronics Conference, Sapporo, Japan, 21-24 jun. 2010, pp 2753-2760. [En línea]. Disponible en: https://www.researchgate.net/publication/233864218_Current_sampling_and_measurement_in_PWM_operated_AC_drives_and_power_converters

[3] L. Corradini, D. Maksimovi, P. Mattavelli y R. Zane, 2015. *Digital Control of High-Frequency Switched-Mode Power Converters* Volumen 48 de IEEE Press Series on Power Engineering. 445 hoes lane piscataway, John Wiley & Sons, 2015.

[4] S. Buso y P. Mattavelli, *Digital Control in Power Electronics*, 1ª Edición, United States of America, Morgan & Claypool, 2006.

[5] J. Rodriguez de Frutos, "Emulador de red mediante un convertidor conmutado diseño del control y de la etapa de potencia", Trabajo fin de grado, Departamento de electrónica, Universidad Carlos III de Madrid, Madrid España, 2016.

[6] Blasco, V.; Kaura, V.; Niewiadomski, W., "Sampling of discontinuous Voltaje and current signals in electrical drives: a system approach", IEEE Trans. On Ind. Appl. Vol. 34, no. 5, Sept.-Oct. 1998, pp. 1123 – 1130.

[7] Zhang, Z. & Chong, K.T. "Second order hold and taylor series based discretization of SISO input time-delay systems", *Journal of Mechanical Science and Technology*, 23 (2009), 136~148, jul. 2009

[8] Powersim Inc. *PSIM User's Guide*, 5ª Edición, Version 10, United States, jun. 2016

[9] Compatibilidad electromagnética (CEM) Parte 3-2: Límites, AENOR UNE-EN 61000-3-2:2015

[10] Directiva 2004/108/CE del Parlamento Europeo y del Consejo, de 15 de diciembre de 2004, relativa a la aproximación de las legislaciones de los Estados miembros en materia de compatibilidad electromagnética y por la que se deroga la Directiva 89/336/CEE.

Apéndice A: Código en C del integrador reiniciado síncrono

```
/*
       Hallo la media de la entrada en el periodo 1/fm y la saco al final del
periodo
*/


#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...

//int gPerMod_i = 0;      //0= no inicializada
static double gPerMod_d = -1.0;  // negativo indica no inicializado

static double sAcum_d;
static double sSigT_d;
static int sCnt_i = 0;

////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
//int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.


       sAcum_d += in[0];
       ++sCnt_i;

             // A ver si cambio de periodo de modulacion
       if(  (sSigT_d -  t) < (t + delt - sSigT_d ) ) {     //Si he saltado de
periodo de modulación
//           printf("\n sTIniPerMod_d = %9.9lf" , sTIniPerMod_d);
             out[0] = sAcum_d / (double)sCnt_i;     // para depu, saco por la de
Vmed2
             sAcum_d =0.0;
             sCnt_i = 0;
             sSigT_d += gPerMod_d;
             }     //if
}


////////////////////////////////////////////////////////////////
```

```
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
             const char *szId, int nInputCount, int nOutputCount,
              int nParameterCount, const char ** pszParameters,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.

       gPerMod_d = 1 / GetPsimValue( reserved_ThreadIndex, reserved_AppPtr, "",
"fm", &nStatus);
//     printf("\n begin gPerMod_d = %lf" , gPerMod_d);
              // Inicializo las stats
       sAcum_d =0.0;
       sCnt_i = 0;
       sSigT_d = gPerMod_d;

}


////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{



}
```

Apéndice B: Código en C del “Second Order Hold” (SOH)

```
/* FOH1 First Order Hold1 y Second Order Hold

      FOH: Entre dos muestreos (marcados por un cambio de valor de la variable
flotante recibida por el puerto 2), saco una rampa temporal con
      origen en la última muestra (valor de la variable flotante recibida por el
puerto 1 y pendiente la de las 2 últimas muestras
      SOH: Saco una parábola con la FOH corregida por la derivada 2a, pero
limitada, por arriba, por la FOH y, por abajo, por el valor anterior (para que no
retroceda)

*/


#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...
double sEps_d = 0.0;      //Epsilon para comparaciones flotantes

//int gPerMod_i = 0;      //0= no inicializada
//bc double gPerMod_d = -1.0;    // negativo indica no inicializado

////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
static double sMues_da[2] = {0.0, 0.0};             //  Muestras anteriores
static double sTiMues_da[2] = {0.0, -1e6};          // Tiempos de esas muestras
static double sDeri1_da[2] = {0.0, 0.0};            // Derivadas primera actual y
anterior
static double sDeri2_da[2] = {0.0, 0.0};            // Derivadas segunda actual y
anterior
double sam_d;
      // ultima orden de muestrear
static double  sPrevSam_d = 0.0;                    // flanco previo
static double  candiSam_d = 0.0;                    // flanco candidato
double At_d;
             // ultimo incremento de t
double nue_d;
```

```
      sam_d = in[1];      //sample del trigger, para comparar
      At_d = t - sTiMues_da[0];

      if( (candiSam_d == sam_d ) && (sam_d <( sPrevSam_d - sEps_d) || sam_d >
(sPrevSam_d + sEps_d) ||               //Si estabilizado el nuevo flanco y: orden
de muestrear (por variación de la var superior a epsilon)
                  (At_d - sTiMues_da[0] > (sTiMues_da[0] - At_d - sTiMues_da[1]
) * 1.1) ) ) {            // o por superación de tiempo anterior

                  // Deslizo muestras y tiempos anteriores
            sMues_da[1] = sMues_da[0];
                  // origino la rampa en el  valor actual de la var de entrada
            sMues_da[0] = in[0];      // Ultima muestra es ésta 0, por si
decidiera tomar más para un second order hold.
            sTiMues_da[1] =  sTiMues_da[0];
            sTiMues_da[0] = t;
                  // Calculo y almaceno ya la primera derivada:
            sDeri1_da[1] = sDeri1_da[0];     // deslizo la anterior
            sDeri1_da[0] = (sMues_da[0] - sMues_da[1] ) / (sTiMues_da[0] -
sTiMues_da[1] );    //primera derivada
            sDeri2_da[1] = sDeri2_da[0];     // deslizo la anterior (aunque aun
no la uso)
            sDeri2_da[0] =  (sDeri1_da[0] - sDeri1_da[1] )  / (sTiMues_da[0] -
sTiMues_da[1] );    //segunda derivada
            out[0] = 1.0;                                                  //
Saco indicativo de cambio
            out[1] = sMues_da[0];                        // Empiezo nueva rampa
por el actual valor.
            out[2] = sMues_da[0];                        // Empiezo nueva rampa
por el actual valor.
            sPrevSam_d = sam_d; // ultima pasa a previa, para detección de
flancos
//printf("\nt=%9.9lf \ty0 = %9.9lf y1 = %9.9lf \tt0 = %9.9lf t1 = %9.9lf \tdy0 =
%9.9lf dy1 = %9.9lf", t, sMues_da[0],  sMues_da[1], sTiMues_da[0], sTiMues_da[1],
sDeri1_da[0], sDeri1_da[1] );
            }
      else {
            candiSam_d = sam_d;                    //Nuevo candidato
            out[0] = 0.0;                                                  //
Saco indicativo de NO cambio
            out[1] = sMues_da[0] + sDeri1_da[0] * At_d;  // FOH: incremento
igual a la primera derivada por el incremento de tiempo.
            nue_d = out[1] + sDeri2_da[0] * At_d * At_d;  // candidata a soh,
con correccion  debida a la derivada 2a. Estrictamente, por taylor, sería la
mitad, pero queda mejor así.
            if( sDeri1_da[0] >=  0.0 ) {           // si subiendo
                  if( nue_d > out[1] ) {                        // si la soh
corregida supera a la foh
                        out[2] = out[1];                              // no la
supero, la igualo.
                        }
                  else {
                        if( nue_d > out[2] ) {    // Si subo respecto a la
anterior
                              out[2] = nue_d;           // subo.
                              }
                        }             //Si no, dejo la anterior.
                  }
            else { // bajando
                  if( nue_d < out[1] ) {                        // si la soh
corregida supera a la foh
                        out[2] = out[1];                              // no la
supero, la igualo.
```

```
                        }
                  else {
                        if( nue_d < out[2] ) {    // Si bajo  respecto a la
anterior
                              out[2] = nue_d;           // bajo.
                              }
                        }              //Si no, dejo la anterior.
                  }
            }
}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
             const char *szId, int nInputCount, int nOutputCount,
              int nParameterCount, const char ** pszParameters,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.

sEps_d = GetPsimValue(reserved_ThreadIndex, reserved_AppPtr, "", "Epsi",
&nStatus);
//    printf("\n begin gPerMod_d = %lf" , gPerMod_d);

}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{

}
```

Apéndice C: Código en C de muestreado deslizante de 1 o 2 muestras, actualizadas en varios instantes

```
/* Una o dos muestras:
       Si 1, en el centro del Tm periodo de modulación
       Si 2, 1 muestra en pico y valle, actualizada cada Tm/2
       Si 4, 2 muestras simétricas respecto al anterior vértice (pico o valle),
actualizadas cada Tm/4. Cada Tm/2, la sola muestra en ese vértice.
       Etc:
*/
#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...

//int gPerMod_i = 0;      //0= no inicializada
static double gPerMod_d = -1.0;  // negativo indica no inicializado

////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
//static double sAcum32_da[5];
static double sSigT_da[7];
static double sTIniPerMod_d = 0.0;
static double sSam_daa[ 6 ][ 32 ];            // array de dim 2, con
almacenamiento de muestras antiguas.
static int sam_ia[ 6 ];                                                  // array
de índice de muestra en el anterior para la siguiente
static int sCnt_i = 0;
static int sIni_i = 0;
long int tiem_l;
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.
int ns_i;
double tEnPerMod_d;
double acu_d;// acumulador para calculos
int i, j;    // indices de fors
int s_i;
#define Adelanto    (-delt)      //p q psim da la i simulada un delt después de
llamarme (creo que es aleatorio a quién llama primero)

       if( ! sIni_i ) {
```

```
            ++sIni_i;    // ya no mas veces
                  // Inicializo las stats
//          sSigT_da[1] = sSigT_da[0];// en el [1], voy a llevar el de 2
muestras: 1 en el pico y otra en el valle.
            for (ns_i = 1, i = 0; i < 6; ++i) {
                  sam_ia[i] = 1;     //Empiezo por la 1, porque la 0 sería al
ppio del Tm
                  for(s_i =  0; s_i < 32; ++s_i) {
                        sSam_daa[i][s_i] = 0.0;   //ini  más fácil que en la
declaración. Todas, aunque no las use.
                        }     // for
                  sSigT_da[i] = gPerMod_d / ns_i;  //  - Adelanto;
                  ns_i *= 2;   // siguiente num de muestras por periodo
                  }     //for
            sSigT_da[0] = gPerMod_d / 2;     // en el [0], voy a llevar el de 1
muestra en el centro (pico) del periodo de modulación. Por eso, desplazo
SigTiempo Tm/2
            sSigT_da[6] = gPerMod_d - Adelanto;    // Basta con adelantar éste,
que es la base.
            sTIniPerMod_d = 0.0;
            }     //if

// if( t>4.9949e-3 && t <4.9951e-3) printf("\n i = %d,  t=%9.9lf \ttEnPerMod_d =
%9.9lf \t sTIniPerMod_d = %9.9lf,\t sSigT_da[0] = %9.9lf, \t delt  = %9.9lf \t
in[0]=%9.9lf" , i, t, tEnPerMod_d, sTIniPerMod_d, sSigT_da[0], delt, in[0] );

      tEnPerMod_d = t - sTIniPerMod_d;
//    printf("\n t = %9.9lf", t);
            // A ver si instantes de muestreo
      for (ns_i = 1, i = 0; i < 6; ++i) {
            if( (sSigT_da[i] - tEnPerMod_d) < ( tEnPerMod_d + delt - sSigT_da[i]
)  ) { // Si estoy lo más próximo al instante de muestreo siguiente
// if(i<2 && t>4.9e-3 && t <5.1e-3) printf("\n i = %d,  t=%9.9lf \ttEnPerMod_d =
%9.9lf \t sTIniPerMod_d = %9.9lf,\t sSigT_da[i] = %9.9lf, \t delt  = %9.9lf \t
in[0]=%9.9lf" , i, t, tEnPerMod_d, sTIniPerMod_d, sSigT_da[i], delt, in[0] );
                  if( i == 0) {
                        out[i] =  in[0];    // saco la muestra a la salida sin
más.
                        }
                  else {
                        sSam_daa[ i ][ sam_ia[i]  ] = in[0];   // guardo  la
muestra
                              //Voy a hallar la media de las 2 últimas muestras
simétricas
                        j = - sam_ia[i];    //que el 0 pase a -ns_i y de ahí,
otra vez a 0, para que dé la media consigo mismo, que es igual a  la última
muestra, que es lo que quiero sacar.
                        while( j < 0 ) {
                              j +=  ns_i;
                              }
                        out[i] =  ( in[0] + sSam_daa[ i ][ j  ])  / 2.0; //
Saco a la patilla la media de esta muestra y su simétrica anterior respecto al
anterior vértice.
//                      if(ns_i == 8)printf("\n t = %9.9lf i = %d j = %d
sam_ia[i] = %d in[0] = %lf sSam_daa[ i ][ sam_ia[i]  ]  = %lf sSam_daa[ i ][ j  ]
= %lf ", t, i, j, sam_ia[i], in[0], sSam_daa[ i ][ sam_ia[i]  ], sSam_daa[ i ][ j
]);
                        if(++sam_ia[i] >= ns_i ) {// actualizo indice a  la
siguiente muestra. Si me paso,
                              sam_ia[i] = 0;      // vuelvo al 0, principio del
array en circulo.
                              }
                        }
```

```
                    sSigT_da[i] += gPerMod_d / (double)ns_i;      //tiempo
previsto del proximo instante de muestreo
//                  if(ns_i == 8) printf("\n t = %9.9lf sSigT_da[i] = %9.9lf", t,
sSigT_da[i]);
                    }     // if
             ns_i *= 2;   // siguiente num de muestras por periodo
             }     //for

             // A ver si cambio de periodo de modulacion
      if(  (sSigT_da[6] -  t) < (t + delt - sSigT_da[6]) ) {     //Si he saltado
de periodo de modulación
             sTIniPerMod_d = t;
//           printf("\n sTIniPerMod_d = %9.9lf" , sTIniPerMod_d);
             for (ns_i = 1, i = 1; i < 6; ++i) {
                    ns_i *= 2;   // siguiente num de muestras por periodo
                    sam_ia[i] = 1;      // no debería ser necesario, pero así no
lo compruebo.. //Empiezo por la 1, porque la 0 sería al ppio del Tm
                    sSigT_da[i] = gPerMod_d / ns_i;
                    }     //for
             sSigT_da[6] += gPerMod_d;
             sSigT_da[0] = gPerMod_d / 2;     // en el [1], voy a llevar el de 1
muestra en el centro (pico) del periodo de modulación.
//           out[0] = sCnt_i;    // para depu, saco por la de Vmed2
//           sCnt_i = 0;
             }     //if
}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
             const char *szId, int nInputCount, int nOutputCount,
              int nParameterCount, const char ** pszParameters,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.

      gPerMod_d = 1 / GetPsimValue( reserved_ThreadIndex, reserved_AppPtr, "",
"fm", &nStatus);
//    printf("\n begin gPerMod_d = %lf" , gPerMod_d);

}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
```

```
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{



}
```

Apéndice D: Código en C de muestreado a fin de periodo de 2 a 32 muestras

```
#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...

//int gPerMod_i = 0;      //0= no inicializada
static double gPerMod_d = -1.0;  // negativo indica no inicializado

////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
static double sAcum32_da[5];
static double sSigT_da[6];
static double sTIniPerMod_d = 0.0;
static int sCnt_i = 0;
static int sIni_i = 0;
long int tiem_l;
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.
int ns_i = 32;
double tEnPerMod_d;
int i;
#define Adelanto   (-delt)

      if( ! sIni_i ) {
            ++sIni_i;    // ya no mas veces
                  // Inicializo las stats
            for (ns_i = 1, i = 0; i < 5; ++i) {
                  ns_i *= 2;   // siguiente num de muestras por periodo
                  sAcum32_da[i] = 0.0;
                  sSigT_da[i] = gPerMod_d / ns_i;
                  }     //for
            sTIniPerMod_d = 0.0;
            sSigT_da[5] = gPerMod_d - Adelanto;
            }     //if

      tEnPerMod_d = t - sTIniPerMod_d;
            // A ver si instantes de muestreo
```

```
      for (ns_i = 1, i = 0; i < 5; ++i) {
            ns_i *= 2;   // siguiente num de muestras por periodo
            if( (sSigT_da[i] - tEnPerMod_d) < ( tEnPerMod_d + delt - sSigT_da[i]
)  ) { // Si estoy lo más próximo al instante de muestreo siguiente
//                printf("\n ns_i = %d,  t=%9.9lf,  tEnPerMod_d = %9.9lf,
sTIniPerMod_d = %9.9lf, sSigT_da[i] = %9.9lf, sCnt_i = %d" , ns_i, t,
tEnPerMod_d, sTIniPerMod_d, sSigT_da[i], sCnt_i );
                  sAcum32_da[i] += in[0];
                  sSigT_da[i] += gPerMod_d / ns_i;
//                ++sCnt_i;
                  }     // if
            }     //for

            // A ver si cambio de periodo de modulacion
      if(  (sSigT_da[5] -  t) < (t + delt - sSigT_da[5]) ) {     //Si he saltado
de periodo de modulación
            sTIniPerMod_d = t;
//          printf("\n sTIniPerMod_d = %9.9lf" , sTIniPerMod_d);
            for (ns_i = 1, i = 0; i < 5; ++i) {
                  ns_i *= 2;   // siguiente num de muestras por periodo
                  out[i] = sAcum32_da[i] / ns_i;   // Lo saco a la patilla
                  sAcum32_da[i] = 0.0;                  // Y lo reinicio;
                  sSigT_da[i] = gPerMod_d / ns_i;
                  }     //for
            sSigT_da[5] += gPerMod_d;
//          out[0] = sCnt_i;    // para depu, saco por la de Vmed2
//          sCnt_i = 0;
            }     //if
}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
            const char *szId, int nInputCount, int nOutputCount,
             int nParameterCount, const char ** pszParameters,
             int *pnError, char * szErrorMsg,
             void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.

      gPerMod_d = 1 / GetPsimValue( reserved_ThreadIndex, reserved_AppPtr, "",
"fm", &nStatus);
//    printf("\n begin gPerMod_d = %lf" , gPerMod_d);

}
```

```
////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{



}
```

Apéndice E: Código en C de muestreado deslizante de las últimas 2 a 32 muestras

```
#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...

//int gPerMod_i = 0;      //0= no inicializada
static double gPerMod_d = -1.0;  // negativo indica no inicializado

////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
static double sAcum32_da[5];
static double sSigT_da[6];
static double sTIniPerMod_d = 0.0;
static double sSam_daa[ 5 ][ 32 ];            // array de dim 2, con
almacenamiento de muestras antiguas.
static int sam_ia[ 5 ];                                                    // array
de índice de muestra en el anterior para la siguiente
static int sCnt_i = 0;
static int sIni_i = 0;
long int tiem_l;
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.
int ns_i;
double tEnPerMod_d;
double acu_d;// acumulador para calculos
int i, j;    // indices de fors
int s_i;
#define Adelanto    (-delt)      //p q psim da la i simulada un delt después de
llamarme (creo que es aleatorio a quién llama primero)

      if( ! sIni_i ) {
            ++sIni_i;    // ya no mas veces
                  // Inicializo las stats
            for (ns_i = 1, i = 0; i < 5; ++i) {
                  ns_i *= 2;   // siguiente num de muestras por periodo
                  sAcum32_da[i] = 0.0;
                  sSigT_da[i] = gPerMod_d / ns_i;
                  for(s_i =  0; s_i < 32; ++s_i) {
```

```
                        sSam_daa[i][s_i] = 0.0;   //ini  más fácil que en la
declaración. Todas, aunque no las use.
                        }     // for
                  }     //for
            sTIniPerMod_d = 0.0;
            sSigT_da[5] = gPerMod_d - Adelanto;    //Basta con ajustar la base.
            }     //if

      tEnPerMod_d = t - sTIniPerMod_d;
            // A ver si instantes de muestreo
      for (ns_i = 1, i = 0; i < 5; ++i) {
            ns_i *= 2;   // siguiente num de muestras por periodo
            if( (sSigT_da[i] - tEnPerMod_d) < ( tEnPerMod_d + delt - sSigT_da[i]
)  ) { // Si estoy lo más próximo al instante de muestreo siguiente
//                printf("\n ns_i = %d,  t=%9.9lf,  tEnPerMod_d = %9.9lf,
sTIniPerMod_d = %9.9lf, sSigT_da[i] = %9.9lf, sCnt_i = %d" , ns_i, t,
tEnPerMod_d, sTIniPerMod_d, sSigT_da[i], sCnt_i );
                  sAcum32_da[ i ] += in[0];
                  sSam_daa[ i ][ sam_ia[i]  ] = in[0];   // tomo la muestra
                        //Voy a hallar la media de todas las últimas ns_i
muestras
                  acu_d = 0.0; // inicializo
                  for(s_i =  sam_ia[ i ], j = 0; j < ns_i; --s_i, ++j) {     //
Empiezo de la última muestra de este grupo y recorro hacia atrás
                        if(s_i < 0) {
                              s_i = 31;    // si rebasado el primero, vuelvo
al último índice
                              }
                        acu_d += sSam_daa[ i ][ s_i ];
                        }     // for
                  out[i] = acu_d / ns_i;    // Lo saco a la patilla
                  if(++sam_ia[i] >= 32) {   // actualizo indice a  la siguiente
muestra. Si me paso,
                        sam_ia[i] = 0;      // vuelvo al 0, principio del array
en circulo.
                        }

                  sSigT_da[i] += gPerMod_d / ns_i; //tiempo  previsto del
proximo instante de muestreo
//                ++sCnt_i;
                  }     // if
            }     //for

            // A ver si cambio de periodo de modulacion
      if(  (sSigT_da[5] -  t) < (t + delt - sSigT_da[5]) ) {     //Si he saltado
de periodo de modulación
            sTIniPerMod_d = t;
//          printf("\n sTIniPerMod_d = %9.9lf" , sTIniPerMod_d);
            for (ns_i = 1, i = 0; i < 5; ++i) {
                  ns_i *= 2;   // siguiente num de muestras por periodo
//                out[i] = sAcum32_da[i] / ns_i;   // Lo saco a la patilla
                  sAcum32_da[i] = 0.0;                    // Y lo reinicio;
                  sSigT_da[i] = gPerMod_d / ns_i;
                  }     //for
            sSigT_da[5] += gPerMod_d;
//          out[0] = sCnt_i;    // para depu, saco por la de Vmed2
//          sCnt_i = 0;
            }     //if
}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
```

```
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
             const char *szId, int nInputCount, int nOutputCount,
              int nParameterCount, const char ** pszParameters,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.

       gPerMod_d = 1 / GetPsimValue( reserved_ThreadIndex, reserved_AppPtr, "",
"fm", &nStatus);
//     printf("\n begin gPerMod_d = %lf" , gPerMod_d);

}


////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{




}
```

Apéndice F: Código en C de cálculo del error cuadrático medio RMS entre dos señales

```
#include <Stdlib.h>
#include <String.h>
#include <math.h>
#include <Psim.h>

// PLACE GLOBAL VARIABLES OR USER FUNCTIONS HERE...
double Pru_d;
////////////////////////////////////////////////////////////////
// FUNCTION: SimulationStep
//   This function runs at every time step.
//double t: (read only) time
//double delt: (read only) time step as in Simulation control
//double *in: (read only) zero based array of input values. in[0] is the first
node, in[1] second input...
//double *out: (write only) zero based array of output values. out[0] is the
first node, out[1] second output...
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationStep(
             double t, double delt, double *in, double *out,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER YOUR CODE HERE...
static int sInid_i = 0;
static int sNElems_i = 0;
static double sPer_d;
static double sRetar_d;
static double sAcum_d;
static double sSigT_d;

       if( !sInid_i ) {
              sInid_i = 1;
              sPer_d = in[2];     // periodo
              sRetar_d =  in[3];  // fase
              sAcum_d = 0.0;      // acumulador
//            printf("\n Ini = %d, sPer_d = %lf, sRetar_d = %lf, Pru_d = %lf " ,
sNElems_i, sPer_d, sRetar_d, Pru_d);
              sSigT_d = sRetar_d;
//            out[0] = 1.0;
//            out[1] = 1.1;
//            out[2] = 1.2;
       }

       if( (sSigT_d - t) < (t + delt - sSigT_d)  ) { //Si he saltado de periodo de
modulación
              sAcum_d += (in[0] - in[1] ) * (in[0] - in[1] );
              ++sNElems_i;
//            printf("\n sNElems_i = %d" , sNElems_i);
              out[0] = sqrt(sAcum_d / sNElems_i);
              out[1] = sNElems_i;
              out[2] = sAcum_d;
              sSigT_d += sPer_d;
```

```
            }     //if
}


////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationBegin
//   Initialization function. This function runs once at the beginning of
simulation
//   For parameter sweep or AC sweep simulation, this function runs at the
beginning of each simulation cycle.
//   Use this function to initialize static or global variables.
//const char *szId: (read only) Name of the C-block
//int nInputCount: (read only) Number of input nodes
//int nOutputCount: (read only) Number of output nodes
//int nParameterCount: (read only) Number of parameters is always zero for C-
Blocks.  Ignore nParameterCount and pszParameters
//int *pnError: (write only)  assign  *pnError = 1;  if there is an error and set
the error message in szErrorMsg
//    strcpy(szErrorMsg, "Error message here...");
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationBegin(
             const char *szId, int nInputCount, int nOutputCount,
              int nParameterCount, const char ** pszParameters,
              int *pnError, char * szErrorMsg,
              void ** reserved_UserData, int reserved_ThreadIndex, void *
reserved_AppPtr)
{
// ENTER INITIALIZATION CODE HERE...
int nStatus = -1; // If the function succeeds, the value of nStatus will be 0.
Pru_d = GetPsimValue(reserved_ThreadIndex, reserved_AppPtr, "", "Per", &nStatus);
       //Vale si pongo una var "Per" en el atributo del subcir

}


////////////////////////////////////////////////////////////////////
// FUNCTION: SimulationEnd
//   Termination function. This function runs once at the end of simulation
//   For parameter sweep or AC sweep simulation, this function runs at the end of
each simulation cycle.
//   Use this function to de-allocate any allocated memory or to save the result
of simulation in an alternate file.
// Ignore all parameters for C-block
// DO NOT CHANGE THE NAME OR PARAMETERS OF THIS FUNCTION
void SimulationEnd(const char *szId, void ** reserved_UserData, int
reserved_ThreadIndex, void * reserved_AppPtr)
{




}
```